





Inhalt

Kap. 0:	Vorwort	3	2.7.1. Methoden einfügen	43	3.10. Wenn gleich nicht gleich gleich ist	95	
0.1.	Dies ist kein fehlendes Handbuch!.....	3	2.8. Keine Ruhe auf den billigen Plätzen!	46	3.10.1. Keine Regel ohne Ausnahme – oder?.....	97	
0.2.	Eine kurze Geschichte über die Struktur.....	5	2.9. Was deins ist, ist auch meins, aber was meins ist, ...	48	3.10.2. Objekt-Apologetik	98	
Kap. 1:	Einführung	7	2.9.1. Und was ist mit den Scope-Popups im Inspector?	49	3.10.3. Haben Sie irgendwelche Referenzen?	99	
1.1.	Xojo ist kein Name und auch kein Getränk.....	7	2.10. Outro	50	3.10.4. Cave leakem!	100	
1.1.1.	Entwicklungssystem.....	7			3.11. Ende des Anfangs.....	102	
1.1.2.	Objektorientiertheit	7					
1.1.3.	Plattformübergreifend	9	Kap. 3:	Einführung in die objektorientierte Programmierung	52	Referenz	104
1.1.4.	Ksojo?.....	9	3.1. OOP, oh weh!	52	Befehle	105	
1.1.5.	Echt kein Getränk?	9	3.1.1. Rant über die Laxheit im Umgang mit der Sprache	54	Datentypen.....	110	
1.2.	liih! Basic!.....	10	3.1.2. Klassenbewusstsein	55	Klassen	140	
1.3.	Installation.....	13	3.1.3. Eventhandler hinzufügen	57	Klassen (allgemein)	142	
1.4.	Die IDE.....	15	3.2. Der Debugger	58	Klassen: Classic-Framework	144	
1.4.1.	Der Project Chooser	15	3.3. Auf die Plätze	61	Klassen: Classic-Framework: Console	145	
1.4.2.	Das Hauptfenster	16	3.3.1. Schaffen wir das?	62	Klassen: Classic-Framework: Desktop	146	
1.5.	Das erste Programm.....	17	3.4. Auf Montage	63	Klassen: Classic-Framework: Desktop: allgemeine Klassen	147	
1.5.1.	Ging schnell, hat funktioniert – was sagt mir das?.	21	3.5. (Mit-)Teilen macht Freu(n)de!	70	Klassen: Classic-Framework: Desktop: Steuerelemente ..	148	
1.6.	If Interesse =	22	3.5.1. Alle für einen!	75			
Kap. 2:	Einführung in die Programmierung	24	3.6. Erben ohne Streit	78	Index	180	
2.1.	Schriftliche Bekanntmachung der Tagesordnung	24	3.6.1. Auf Identitätssuche	79			
2.1.1.	Und wenn ich mehr als ganze Zahlen will?	26	3.6.2. Geworfene Typen	80	Impressum	185	
2.1.2.	Negative Stimmung?	28	3.6.3. Das ist ja super!	80			
2.2.	Nun aber zur Tagesordnung!.....	29	3.6.4. ... und noch superer!	81			
2.3.	Ferdammte Vehler?	33	3.6.5. Eigentümlichkeiten von Eigenschaften	82			
2.4.	Sei nicht immer so berechnend!.....	34	3.6.6. Zusammenfassender Ausblick	82			
2.5.	Nun hör mir doch mal zu!	37	3.7. Geschwätzige Ereignisse	84			
2.5.1.	String vs. Text.....	37	3.8. Klasse sucht Anschluss	87			
2.6.	Jetzt geht's rund!	39	3.8.1. Eilnachricht für	91			
2.7.	Programmieren mit Methode(n)	42	3.8.2. ... eigene Klassen?	92			
			3.9. Stunde der Entscheidung	94			



KAP. 0: Vorwort

0.1. Dies ist kein fehlendes Handbuch!

Was aber dann?

Nun, hallo zunächst, und lassen Sie mich gleich mal abschweifen, damit Sie sich an meine schriftliche Geschwätzigkeit gewöhnen können:

Seit 2014 darf ich mich als deutschen Entwicklerevangelisten für Xojo bezeichnen. Xojo, das ist dieses wirklich extrem einfach zu lernende und immer vielseitiger mutierende Programm zum Programmieren – fachsprachlich Entwicklungssystem –, um das es im Folgenden gehen soll.

Ob „Evangelist“ in den heutigen Zeiten eine wirklich so geschickte Berufsbezeichnung ist, lass ich mal dahingestellt. Apple hat damit angefangen, personifiziert durch einen Herrn mit dem schnittigen Namen [Guy Kawasaki](#). Dieser trug seine Begeisterung für ein kleines Underdog-Betriebssystem mit Namen Mac OS und die damit verbundenen Rechner in die Programmierer-Community und machte es nicht durch übertriebenes Marketing-Tamtam, sondern durch Überzeugungsarbeit in Form von Workshops, Schulungen und Büchern ein ganzes Stück populärer.

In den politisch noch so erfrischend unkorrekteren Zeiten war dann der Name schnell gefunden: Ein jeder, der die frohe Botschaft einer technologischen Innovation in die Welt trägt, solle sich fortan Evangelist nennen.

Sie dürfen mich gerne mal auf meine philosophische Weltsicht ansprechen, wenn Sie Lust auf einen stundenlangen ad hoc-Vortrag verspüren sollten. Ich kann Ihnen die Essenz aber auch in aller Kürze auf die Nase binden: Ich bin zutiefst davon überzeugt, dass in einem komplexen System wie unser aller Leben jede Lösung, die stets nur einen Pol berücksichtigt und den oder die scheinbaren Widerparts ignoriert, wirklich keinerlei dauerhaften Nutzen bringt.

Ich werde daher niemals so tun, als sei Xojo das alleinseligmachende Entwicklungswerkzeug und alle anderen zu verdammten. Im Gegenteil: Manchmal ist es einfach notwendig, selbst bei einer Xojo-Lösung Funktionen oder Bibliotheken, die mit anderen Werkzeugen erstellt wurden, zu integrieren. Manchmal muss so etwas gar selbst erstellt werden.

Aber auch in diesen Fällen kann Xojo eine Brücke zum Verstehen anderer Programmiersprachen sein. Bestimmte Strukturen und Verfahrensweisen finden sich allüberall, nur sind die Programmiersprachkommandos und der „Satzbau“ – die Syntax der Programmiersprachen – unterschiedlich schwer zu lernen und zu lesen.

Bei Xojo bewegen wir uns zumeist in fast klar lesbarem Englisch, gepaart mit einer sehr übersichtlichen Syntax. Ich denke, es bedarf keiner Erläuterung, dass der Einstieg in eine solcherart gestaltete Programmiersprache einfacher fällt als in eine, die mehr oder weniger aus kryptischen Abkürzungen, unübersicht-

lichen Klammerebenen und Ähnlichem besteht. Wohlgemerkt: Nichts gegen solche Sprachen! Ich kann allerdings aus eigener Erfahrung bestätigen, dass Ihre Bezwigung zur immensen Herausforderung werden kann. Mittlerweile laufen meine ersten C-Programme auf einem Arduino, den ich mittels Xojo vom Raspberry Pi anspreche. Dass das im fortgeschrittenen Alter¹ doch noch geklappt hat, kann ich eindeutig auf die Beschäftigung mit Xojo zurückführen. Die seinerzeit mal erstandenen Objective C-Bücher, die mir trotzdem nicht zur Beherrschung von Apples XCode-Entwicklungssystem halfen, habe ich mittlerweile in die hinterste Regalecke geschickt, da ich sie nicht mehr benötige.

Ja nun, jedenfalls erreichen mich häufiger Fragen der Art, wann denn eine deutsche Version des umfangreichen Xojo-Lehrbuchs zur Verfügung stünde. Und diese Fragen schmettere ich in der Regel ab, denn das Ding ist ein monumentales Werk! Alleine eine reine Übersetzung würde schon Zeit- und damit finanzielle Ressourcen verschlingen, über die ich nicht verfüge. Kommt noch dazu: Xojo befindet sich seit Einführung der iOS-Unterstützung in der Transformation zum neuen Xojo-Framework, das viele alte Zöpfe abschneidet und durch moderne Features ersetzt. Dies steht auf den anderen Plattformen aber noch nicht vollständig zur Verfügung, wäre aber das Mittel der Wahl, um die zeitgemäße Xojo-Programmierung zu erklären. Die eng-

¹ in Heinz-Strunk-Definition



lischen Handbücher selbst mit ihren Hunderten von Seiten erklären aber größtenteils noch die alten Features, die zwar noch auf lange Zeit verfügbar sein werden, m. E. aber nicht mehr Zentrum der Programmierung darstellen sollten. Aus dem Nachbessern des übersetzten Handbuchs, wenn es denn überhaupt schaffbar wäre, käme man kaum raus, wollte man mit Xojos vierteljährlichem Releasezyklus mithalten.

Fazit also: Auf zumindest mittlere Sicht wird es kein offizielles deutsches Xojo-Handbuch geben. Und, mal ganz ehrlich: Gedruckte Bücher und reine PDFs sind so 20. Jahrhundert! Und das sage ich, der ich seit über 20 Jahren in der Druckvorstufe tätig bin! Wobei ich den gedruckten Roman oder das Magazin jeder digitalen Ausgabe vorziehe. Fakt ist aber doch: Ein Lehrbuch sollte auch als Nachschlagewerk dienen. Und wenn man sich mit Programmierung beschäftigt, ist der Computer ja nicht weit, der viel schneller nachschlagen kann als der Mensch. Und der nicht nur Text, sondern auch bewegte Bilder, Töne und Daten transportieren kann.

Deshalb wird dies hier kein vollständiges Handbuch werden und auch nicht in gedruckter Form vor Ihnen landen. Stattdessen aber ein erweiterbares eBook/interaktives PDF, wobei sein Inhalt zu einem Gutteil von Ihrem Feedback abhängt. Und von den Videos, die ich zwischenzeitlich erstelle und ebenso hier verlinken werde. Und natürlich von den vorher erwähnten häufig knappen Ressourcen.

Ich möchte Sie daher zu reichlichem Feedback auffordern. Vielleicht gibt es sogar einen Teilaspekt, zu dem Sie selbst ein Kapitel verfassen können. Denn wenn es einen Aspekt von Xojo gibt, den ich jenseits der beschriebenen Vorzüge hervorheben würde, dann die extrem hilfsbereite und kooperationsfreudige [Xojo-Anwendergemeinschaft](#).



0.2. Eine kurze Geschichte über die Struktur

Großes Problem: Welchen Kenntnisstand kann ich bei Ihnen voraussetzen? Idealerweise sollte ein kompletter Programmereinsteiger sich nach einigen Kapiteln als kompetenter Programmierer in die Welt begeben können. Bis dahin haben Quereinsteiger mit bereits vorhandenem Hintergrundwissen aber vielleicht schon gelangweilt abgeschaltet.

Daher folgender Vorschlag: Folgen Sie den Farben! Und Ihrem Interesse!

Die Grundstruktur folgt einem hoffentlich nachvollziehbaren Lehrpfad für den Einsteiger. Dieser ist **grün** markiert, so wie diese Seite. Ich bemühe mich auf den grünen Seiten, nicht mit Fachbegriffen um mich zu schmeißen, damit sich der Leser nicht in zu viel Vokabelnachschlagen verstrickt.

Neue Begriffe werden in den grünen Bereichen erklärt und sollten sich auch im Index wiederfinden.

Die **orangefarbenen** Seiten richten sich an den Leser mit Vorwissen. Hier werden die grünen Abschnitte vertieft und fachspezifischer erläutert. Wenn Sie, sollten Sie den Einsteigerpfad gewählt haben, das Gefühl haben, die Grünschnäbeligkeit zu verlieren, wäre es also durchaus empfehlenswert, zu früheren Kapiteln zurückzukehren, sie vielleicht noch einmal zu überfliegen und dann in den vertiefenden Teil einzusteigen.

Für alle gleichermaßen relevant sind dann noch die in gediegenem **Violett** daherkommenden Seiten. Hier finden Sie Über-

blicke, Referenzen, Listen und Tabellen – alles, was man fürs schnelle Nachschlagen benötigt.

Die Typografie ist bewusst einfach gehalten:

Neue Fachbegriffe und Definitionen

finden Sie grün hinterlegt eingeleitet, und

Programmcode

steht auf grauem Grund.

Ein ↵ in Codezeilen bedeutet, dass Sie hier keine Zeilenschaltung eingeben sollten – also nicht Return drücken. Eine Code wie

```
MsgBox "Drücken Sie diesen Knopf nicht noch ↵
einmal!"
```

gehört also in eine Zeile, was aber aus Satzgründen nicht ging.

Und wenn mir mehr einfällt, wird dieser Teil einfach ergänzt – schauen Sie bei „Neuauflagen“ also ruhig auch in schon gelesene Bereiche hinein!

Ganz wichtig aber noch: Lassen Sie sich durch die Kapitelreihe nichts aufzwingen! Ich versuche zwar, einen nachvollziehbaren Weg einzuhalten, aber nicht alles, was ich erzähle, ist von gleicher Wichtigkeit.

Als schnelle Wegweiser dafür sind die folgenden Symbole gedacht:

Randnotizen sind ergänzende Erklärungen, Hintergrundinformationen und Geschichten – nicht wirklich völlig relevant und manchmal auch nur Beiwerk.

Wichtige Theorie wird durch eine Büroklammer gekennzeichnet.

Praxisabschnitte tragen einen Computer als Erkennungssymbol.

Plattformdetails, die nur für eine bestimmte Plattform gelten, erkennen Sie am Symbol einer Teilmenge. Sie finden die jeweilige Plattform hervorgehoben als Klartext daneben.

Wichtige Hinweise besitzen einen Schlüssel als Markierung. Hier finden Sie also die Schlüssel zur Lösung mancher üblichen Probleme.



EINFÜHRUNG

Schnell Xojo kennenlernen! Hier finden Sie Wissenswertes über

- **Xojo: Geschichte, Struktur, Entwicklung der Sprache**
- **den Start: Installation, Aufbau der IDE**
- **Beispielprojekte: Desktop**



KAP. 1: Einführung

1. Xojo ist kein Name und auch kein Getränk



Wenn ein Buch schon mit zwei Fehlbehauptungen eröffnet, ist es höchste Zeit, es sofort wieder zuzuklappen. Bzw. es zu schließen, die Datei zu löschen und sich seriöseren Dingen zuzuwenden, oder?

Nun, selbstverständlich ist Xojo ein Name. Aber auch ein Akronym – soll heißen: ein Abkürzungskunstwort mit Sinn dahinter –, und diesen zu kennen, hilft schon ein wenig, sich mit Xojo anzufreunden. Die Heimat von Xojo befindet sich in den USA, weshalb der Sinn sich auch nur mit aus dem Sächsischen gemannten Kenntnissen erklärt: Der Name setzt sich zusammen aus einigen aus dem Begriff

Xplatform Object Oriented (Development System)

zusammengeklauten Buchstaben, den Sie relativ quellgetreu mit

Plattformübergreifendes Objektorientiertes (Entwicklungssystem)

übersetzen können.

Was das heißt? Rollen wir den Begriff mal von hinten auf:

1.1.1. Entwicklungssystem

Im Gegensatz zu einem offenen Baukasten-System, bei dem Sie sich alle Bestandteile aus verschiedenen einzelnen Komponenten zusammensuchen müssen – etwa einen Editor für den Quelltext, einen Layout-Editor zum Design von Fenstern und ähnlichen Programmelementen, einen Compiler zum Übersetzen Ihres Quelltextes usw. –, bekommen Sie mit Xojo alles geliefert, was Sie zum Programmieren benötigen. Und dies ist alles in einem einzigen Programm integriert, weshalb Xojo ganz der Vollständigkeit halber auch als

integriertes Entwicklungssystem

bezeichnet werden kann. Dem Programmiererhang zum Benutzen englischer Begriffe geschuldet ist dann auch die gebräuchliche Abkürzung

IDE = Integrated Development Environment

womit das „System“ durch „Umgebung“ ersetzt wurde, um klarzustellen, dass man eben kein Baukasten-, sondern ein All-Inclusive-System benutzt.

In unserem Fall heißt das, dass Sie in der Regel ein einziges Fenster von Xojo zu Gesicht bekommen, ganz ungeachtet der Programmieraktivität, der Sie sich gerade hingeben.

1.1.2. Objektorientiertheit

Vielleicht kennen Sie die alte Analogie, in der das Programmieren mit einem Kochrezept verglichen wird. Die dürfen Sie auch gerne weiterbenutzen, nur sind wir jetzt im 21. Jahrhundert angekommen, und unsere Küchengeräte sind viel intelligenter als anno dunnemals.

Aber auch heute noch fangen die Rezepte mit einem „Man nehme ...“ an. Na gut, noch eine Unwahrheit: So altmodisch sind die wenigsten. Eher steht da jetzt „Zutaten“.

Beim Programmieren ist es ganz ähnlich: Sie müssen erst einmal festlegen, womit Sie arbeiten wollen: Mit einer Zahl (und falls ja: Nur eine Ganzzahl, oder darf sie Kommastellen beinhalten? Bleibt sie positiv oder darf sie ins Minus rutschen?)? Mit einem Text, mit Bildern, Dateien, Datenbanken ... wie auch immer: Sie sagen dem Computer, was die Bestandteile Ihres „Rezepts“ sind.

Und dann arbeitet er brav alle Schritte in Reihenfolge ab, so wie sie es ihm Schritt für Schritt erklären. Das wäre das lineare Programmieren, so wie es auch heute noch in einigen Sprachen und Teilbereichen Anwendung findet.

Wenn sich aber Gelegenheit ergeben sollte, dass Sie sich einmal selbst beim Kochen nach Rezept beobachten können, dann werden Sie ziemlich wahrscheinlich feststellen, dass Sie die



Reihenfolge gar nicht sklavisch einhalten. Viel wahrscheinlicher werfen Sie ein paar Zutaten in den Mixer und lassen ihn diese gründlich verrühren, während Sie schon einmal den Backofen vorheizen und schnell nochmal die Spülmaschine anwerfen. Alle Achtung! Sie betreiben hier aber mächtig

Multitasking!

Sie erledigen mehrere Dinge quasi gleichzeitig. Naja, streng genommen nicht wirklich, aber Sie besitzen so schlaue Geräte, dass Sie diese einfach instruieren können und drauf vertrauen, dass sie ihren Job machen. Sind sie damit fertig, melden sie sich eventuell mit einem Ping! zurück und warten auf neue Anweisungen.

Aber das Rezept war jetzt nur eine ungefähre Richtlinie, um keinen Schritt zu vergessen. Wenn der Käse auf der Quiche gesunde Bräune angenommen hat, wollen Sie diese vermutlich aus dem Backofen befreien, auch wenn das ein eigentlich noch ferner Punkt auf der Rezeptliste ist. Ich würde es jedenfalls empfehlen, auch wenn dieser Rat eher aus der Beobachtung denn aus eigener Kocherfahrung stammt.

Wie würde ein solches Rezept aussehen müssen? Es müsste einzelne Blöcke enthalten, die die jeweiligen Küchengeräte adressieren, so in etwa

Backofen: Heize auf 180° vor und gib mir Bescheid!

Mixer: Quirl alles zusammen, bis eine sämige Konsistenz erreicht ist. Stell dich dann aus!

Die sklavische Reihenfolge des linearen Rezepts weicht also einer flexiblen Handlungsanweisung, die in Betracht zieht, dass

Sie in Ihrer Küche Dirigent und Regisseur sind, bei vielen Routinehandlungen aber auf ein eingespieltes Ensemble vertrauen können.

Die Entwicklung von Brotbackautomaten und Multifunktionsküchengeräten ein kleines bisschen in die Zukunft projektiert zeichnet ein noch komfortableres Bild: Wie wäre es, wenn die Geräte einfach miteinander kommunizieren und womöglich eine Küchendrohne die restlichen manuellen Arbeiten erledigt? (Diese Vision müssen Sie jetzt nicht unbedingt gut finden. Einfach mal so als Gedankenspiel, ok?)

Dann würde der Backofen Bescheid geben, wenn er die richtige Temperatur hat, und der künstliche Küchenknecht bestückt ihn mit den vorbereiteten Zutaten in den geeigneten Behältnissen ...

Wenn Sie diese Idee nachvollziehen können, dann haben Sie das Prinzip von objektorientierter Programmierung verstanden: Sie geben keine feste Ablaufreihenfolge vor. Vielmehr sagen Sie der Schaltfläche auf einem Fenster (gerne als „Button“ oder „Knopf“ bezeichnet), was passieren soll, wenn er gedrückt wird, oder Sie bringen einem Textfeld irgendwelche Tricks bei, die es bei Texteingabe aufführen soll.

Auch wenn es natürlich immer noch bestimmte Reihenfolgen gibt, die beachtet werden, und auch wenn in Wirklichkeit gar nicht alles gleichzeitig,

sondern nur scheinbar so erledigt werden mag: Sie können so die Intelligenz der Programmelemente nutzen, ohne ihnen alles von der Pike auf beibringen zu müssen.

Während ich selbst ein paar Argumente gegen zu starke Automatisierung in der Küche vorbringen könnte: Beim Programmieren mit modernen Betriebssystemen fällt mir keines ein, das gegen den Komfort spricht, den ebenjene Betriebssysteme insbesondere in der Kombination mit objektorientierter Programmierung mit sich bringen.



Abb. 1: Die Welt ist schon so oft begrüßt worden, aber eine bessere ist sie trotz alledem immer noch nicht ...



1.1.3. Plattformübergreifend

Im Computersinne ist eine Plattform die Kombination von Hard- und Software, auf der eine Anwendung läuft. Xojo unterstützt zurzeit die Desktop-Betriebssysteme macOS, Windows und Linux – es läuft auf diesen Plattformen, und man kann damit für diese Plattformen entwickeln.

Man könnte also sagen, dass Xojo im doppelten Sinne plattformübergreifend befähigt ist:

- ▶ Egal, welches dieser Betriebssysteme Ihre bevorzugte Entwicklungsplattform darstellt: Xojo ist auf allen davon mit identischer Funktionalität (und einer Einschränkung, s.u.) verfügbar.
- ▶ Sie können von jeder dieser Plattformen für jede dieser Plattformen entwickeln, oder auch für alle drei zugleich(!)

Dann gibt es noch weitere Plattformen, für die entwickelt werden kann, die aber keine Xojo-Entwicklungsplattform darstellen:

- ▶ Web-Anwendungen, die auf den Webservern der jeweiligen Desktop-Plattformen laufen,
- ▶ Kommandozeilen-Anwendungen als spezielle Unterart der Desktop-Plattformen: Programme, die ohne graphische Benutzeroberfläche funktionieren, also z.B. Hintergrundprozesse oder Programme, die aus dem Terminal heraus, also per Kommandozeileneingabe, gestartet werden,
- ▶ Raspberry Pi 2 & 3 als Unterart eines Linux-Rechners mit einer ARM-CPU
- ▶ und schließlich iOS in Form von iPad- und iPhone-Anwendungen. Hier liegt die einzige Einschränkung vor, denn für die iOS-Entwicklung ist ein Mac als Entwicklungsplattform zwingende Vorbedingung. Grund dafür ist die Existenz des iOS-Simulators als reine macOS-Anwendung. Ohne diesen müsste man theoretisch jedes Programm zum Testen auf ein

iOS-Gerät transferieren, dort starten, hätte kein vernünftiges Debugger-Feedback (soll heißen: Es wäre unglaublich aufwendig, Programmfehlern auf die Spur zu kommen) ... nee-nee, das würde keinen Spaß machen.

1.1.4. Ksojo?

Niemand wird Sie steinigen, wenn das Programm bei Ihnen nach Aborigine-Dialekt klingt. Die offizielle Aussprache allerdings ist eher wie (Verzeihen Sie mir, wenn ich nicht die offizielle Lautschreibungs-Nomenklatur verwende):

Szohjoe

also mit scharfem S wie im englischen „So brilliant!“ und zweimal betontem O. Das alles aber nur, weil ich häufig danach gefragt werde ...

1.1.5. Echt kein Getränk?

Doch, ist es auch. Es besteht aber kein Verwandtschaftsverhältnis zwischen dem Entwicklungssystem und einem amerikanischen Sportgetränke- und -Nahrungsmittelergänzungshersteller. Außerdem schreibt sich dieser in Versalien, und daher ist die Abschnittsüberschrift dann doch nicht ganz so doppelt gelogen wie anfänglich behauptet.



1.2. liih! Basic!



Die Wahl einer Programmiersprache oder eines ganzen Entwicklungssystems ist immer auch eine emotionale Angelegenheit. Man will nicht heute womöglich Geld und eine erkleckliche Menge Einarbeitungszeit investieren und sich morgen dank Insolvenz des Herstellers oder, heutzutage noch häufiger, durch irrsinnige Monopolspielchen an der Börse nach einer neuen Lösung umsehen müssen.

Insofern mag es Sie beruhigen, dass Xojo bald 20. Geburtstag feiern kann, dass es sich im behüteten Privatbesitz befindet und Ideen, die der schnellen Monetarisierung dienen, von Seiten der Geschäftsführung keinerlei Sympathie entgegengebracht wird.

Es mag Sie aber auch irritieren, falls Sie bisher noch gar nichts von Xojo gehört haben sollten. Das mag an seiner Entwicklungs geschichte liegen und eventuell auch daran, dass die Entscheidung für oder gegen eine Programmiersprache viel stärker von Emotionen geleitet wird, als es der Eröffnungssatz vermuten lässt. Kleiner Ausflug in die Firmenhisto rie:

1997 veröffentlichte der US-Amerikaner Andrew Barry die Shareware CrossBasic für den Apple Macintosh, damals als reine Mac-Anwendung. Den plattformübergreifenden Namen erhielt es daher, dass es sowohl native Mac- als auch Java-Programme erzeugen konnte, und, tja nun, weil seine Programmiersprache aus einem Basic-Dialekt entwickelt wurde.

Und was Basic angeht, hat diese Sprache in den Frühzeiten einen schlechten Ruf bekommen, den sie seither nicht mehr los wurde.

Basic

als solches steht für

Beginner's All Purpose Symbolic Instruction Code,

also soviel wie „Allzweckprogrammiersprache für Anfänger“.

Und, zugegeben, das war Basic auch. Anfänglich. Es war eine interpretierte Programmiersprache, soll heißen, es wurde niemals echter Maschinencode erzeugt, sondern ein Übersetzungsprogramm erledigte dies Zeile für Zeile während der Laufzeit. Klarer Fall, dass das sich insbesondere bei den nicht wirklich rasanten CPUs der damaligen Zeit negativ auf die Performance auswirkte.

Der Code selbst war zwar fein lesbar, da sich die Befehlsworte aus Allerwelts-Englisch ableiteten und wenig bis keine kompliziert zu merkenden Abkürzungen oder komplexe Klammerstrukturen beinhaltete.

Er war allerdings auch schnell extrem unübersichtlich, da man hier – zumindest in den frühen Tagen – fröhlich per Anwahl einer Zeilennummer im Code hin- und herhüpfe, wild und frei Speicherbereiche lesen und beschreiben konnte ... Sie haben vielleicht mal den Begriff „Spaghetticode“ für eine unübersichtliche ellenlange Ansammlung von Programmzeilen gehört. Das kann man mit schlechtem Programmierstil immer noch, aber Basic lud sicherlich dazu ein. Es war für heutige objektorientierte Programmierung somit extrem ungeeignet!

Im Laufe der Zeit entwickelte sich Basic aber weiter. Schon zu Heimcomputertagen entstanden sehr respektable Basic-basierte Entwicklungssysteme, die echten Maschinencode erzeugen und sehr strukturierte Programmierung zulassen konnten.

So verließ man sich auch bei FYI Software, der Firma, die die Rechte an CrossBasic kurze Zeit später erwarb (und die heute unter den Namen Xojo, Inc. immer noch für die laufende Weiterentwicklung des Systems verantwortlich zeichnet), auf die Innovationsfreudigkeit und Aufgeschlossenheit der Entwickler, als man es unter dem Namen REALbasic neu herausbrachte (der alte Name war bereits anderweitig vergeben) und später, als Windows und Linux als neue Zielplattformen dazugekommen waren (und nicht zuletzt, weil die Ähnlichkeiten zu Basic immer stärker geschrumpft waren) als Real Studio neu deklarierte.

Erstaunlicherweise aber sind Programmierer oftmals eherne Traditionalisten, und die tiefverwurzelten Vorurteile gegen alles, was irgendwie mit Basic zu tun hatte, lassen selbst heute noch manchmal Stimmen laut werden, die Xojo die ernsthafte Befähigung zur Programmiersprache absprechen. Sollte Ihnen das aus dem Bekanntenkreis passieren, ein ernstgemeinter Vorschlag: Lassen Sie sich einmal die Vorbehalte erläutern. Nicht, dass Xojo perfekt wäre: Es gibt noch eine Vielzahl von Verbesserungsmöglichkeiten. In den allerallermeisten Fällen aber werden Sie feststellen, dass sich diese Kritik überhaupt nicht an der Realität Xojos ausrichtet.

Noch erstaunlicherweise waren die Vorbehalte gegen Basic-Dialekte plattformabhängig. Unter Windows erfreute sich



Visual Basic sehr großer Beliebtheit, bevor es dort durch die Einführung des .Net-Frameworks viele seiner Anhänger verlor. Diese freut es in der Regel zu hören, dass Xojo einen Migrationsassistenten für die Übernahme von VB-Projekten in Xojo bereitstellt.

Womöglich war es der alte Anfänger-Anspruch von Basic, der „ernsthafte“ Programmierer von einer ernsthaften Auseinandersetzung mit früheren Versionen abhielt.

Nun, heute erfreut sich Python großer Beliebtheit, und neben Apples neuer Sprache Swift – sehr grob gesagt ein modernisierter C-Dialekt – werden gerne alle möglichen Abarten von C verwendet.

Während C in nativen Maschinencode kompiliert wird, wird Python häufig interpretiert – ganz wie Basic seinerzeit. Wobei die Interpreter heute freilich flinker sind als zu 8 Bit-Tagen.

Bei genauerer Betrachtung finden Sie aber viele Ähnlichkeiten zwischen Python-Code und dem von Xojo. Nur ist ersterer abgekürzter und benötigt entsprechend ein Stück mehr Gedächtnisleistung. Was man dann bevorzugt, ist ganz individuell zu klären. Beurteilen Sie aber gerne einmal selbst anhand dieser von Wikipedia ausgeliehenen und ergänzten Gegenüberstellung, welchen Code Sie besser lesen können. Es geht jedesmal um die Berechnung der [Fakultät](#), und nur um's noch einmal zu betonen; nicht ein „Besser oder Schlechter“ soll hier angestrebt werden!

C:

```
int fakultaet(int x) {
    if (x > 1)
        return x * fakultaet(x - 1);
    else
        return 1;
}
```

Python:

```
def fakultaet(x):
    if x > 1:
        return x * fakultaet(x - 1)
    else:
        return 1
```

Xojo:

```
Function Fakultät(X as Integer) As Integer
    if (x > 1) Then
        return x * Fakultät(x - 1)
    else
        return 1
    end if
End Function
```

Sollten Sie jetzt feststellen, dass Sie in Xojo mehr zu tippen haben: Nur marginal. Einerseits hilft Ihnen die Autocomplete-Funktion von Xojos Code-Editor bei der Eingabe von Befehlen und Strukturen, und andererseits geben Sie die erste und letzte Zeile gar nicht ein, sondern erstellen Sie grafisch gestützt als Programmobjekt. Der Vorteil: Sie müssen sich nicht auf der Suche nach einer Funktion durch ein langes Listing scrollen, sondern haben die Funktion als sichtbares Objekt zum Schnellzugriff parat. Mit Minimalfenstergröße sieht das so aus wie auf der Folgeseite abgebildet:



Auch die im [Wikipedia-Artikel](#) erwähnte einzeilige Formulierung ist in Xojo möglich:

C:

```
int fakultaet(int x) {
    return (x > 1) ? (x * fakultaet(x - 1)) : 1;
}
```

Python:

```
def fakultaet(x):
    return x * fakultaet(x - 1) if x > 1 else 1
```

Xojo:

```
Function Fakultät(X as Integer) As Integer
    return if (X > 1, x * fakultät(X - 1), 1)
End Function
```

Wobei die Zeile in der Mitte die einzige wäre, die analog zum Bild rechts im Code-Editor einzutragen wäre. Und ja: In Xojo können Sie Umlaute verwenden, und Groß- oder Kleinschreibung werden nicht beachtet.

Am Ende, wenn Sie ein Programm erzeugen, wird der gleiche Compiler angeworfen, der u. a. auch bei Apples [XCode](#) werkelt: [LLVM](#). Abgesehen von den Optimierungseinstellungen für diesen Compiler, die bei Xojo zurzeit noch nicht benutzerdefinierbar sind, wird sich der erzeugte Code (von ein paar Ausnahmen abgesehen, auf die ich später noch eingehen werde) also nicht großartig voneinander unterscheiden – auch hinsichtlich der Performance.

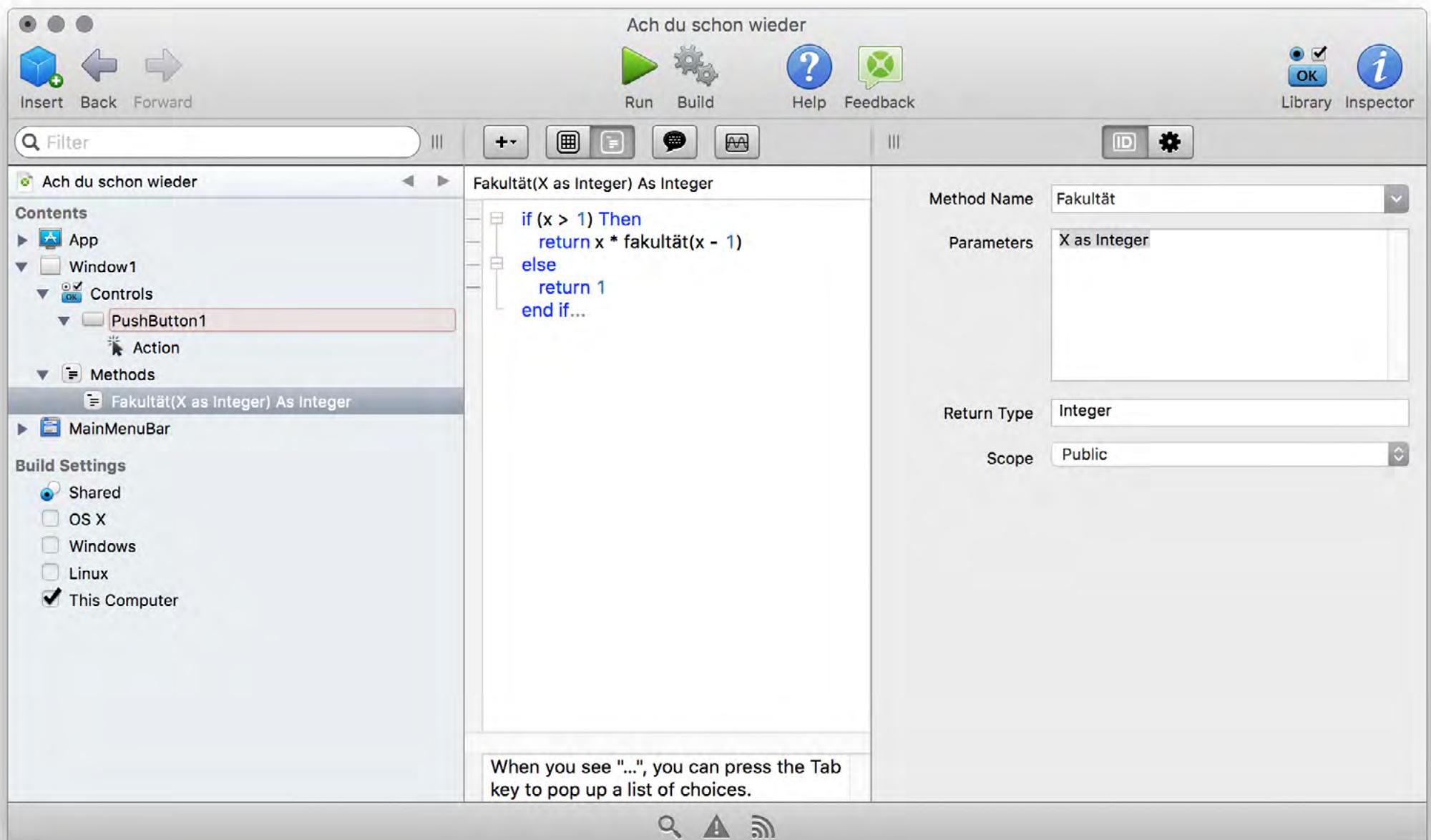


Abb. 2: Die Fakultäts-Funktion in Xojo

Um ein Fazit zu ziehen: **Xojo ist nicht Basic.**

Und

Xojo ist eine moderne, objektorientierte, punktnotierte Programmiersprache mit vielseitigen Einsatzmöglichkeiten, die sich aufgrund ihrer Einfachheit auch sehr gut zum Programmierenlernen eignet.

Xojo ist gleichermaßen das Entwicklungssystem, das für die Programmierung in der Sprache Xojo geschrieben wurde.

Übrigens in Xojo!



1.3. Installation



Binsenweisheit: Um mit Xojo zu arbeiten, sollten Sie es zunächst einmal auf Ihrem Arbeitsrechner installieren. Gehen Sie dazu einfach im Webbrowser auf die Seite

<http://xojo.com/download/>

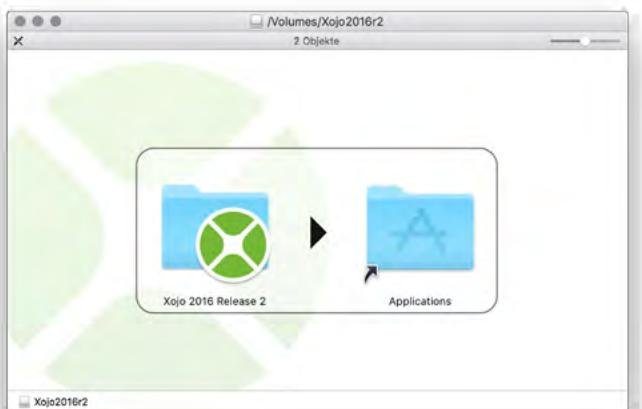
Hier werden Sie zunächst gebeten, ein Konto einzurichten. Dabei müssen Sie keine Kontoinformationen oder weitere sensible Daten eingeben – die hier von Ihnen definierte Benutzername/Passwortkombination ist aber gleichzeitig Ihr Schlüsselbund für eventuell einmal gekaufte Lizenzen und bietet Ihnen die Möglichkeit, diese auch zu verwalten.

Gleichzeitig ist es Ihre Eintrittskarte in das [Xojo-Benutzerforum](#), das ich bestimmt noch ein paarmal völlig zurecht über den grünen Klee loben werde. Nicht selten bekommt man hier Antworten auf brennende Programmierfragen nach nur wenigen Minuten, und mit derzeit über 15.000 Mitgliedern und einer Viertelmillion Beiträgen stellt es ein immenses Archiv bereits beantworteter Anwenderfragen dar.

Haben Sie ein Konto angelegt, sollte die Seite ähnlich wie hier aussehen. Klicken Sie einfach auf den Download-Button des Betriebssystems, das Sie als Entwicklungsplattform verwenden möchten. Nach einiger Zeit – Sie müssen mit irgendwas um 500 MB an Daten rechnen – haben Sie dann eine Installationsdatei, die sich je nach Plattform ein wenig unterschiedlich installieren lässt:

The screenshot shows the Xojo download page. At the top, there's a banner for 'HiDPI-Unterstützung für Mac & Windows' with a 'Download' button. Below the banner, the Xojo logo is displayed. The main content area features a large green 'XOJO' logo. To its right, the text 'Xojo 2016r1.1 herunterladen' is prominently displayed. A detailed description follows, mentioning that Xojo allows users to develop their own apps across various platforms like desktop, web, iOS, and Raspberry Pi. Below this, a 'Download für OS X' button is visible. Further down, sections for 'Andere Plattformen' (OS X, Windows, Linux, Red Hat Linux) and 'Neuer Xojo-Anwender?' are shown. At the bottom, there's a footer with links to Xojo resources, developer links, and company information, along with social media icons and a language selection dropdown.

macOS: Doppelklicken Sie die DMG-Datei. Ziehen Sie dann den Xojo-Ordner im Finder auf das Symbol für das Programmverzeichnis.

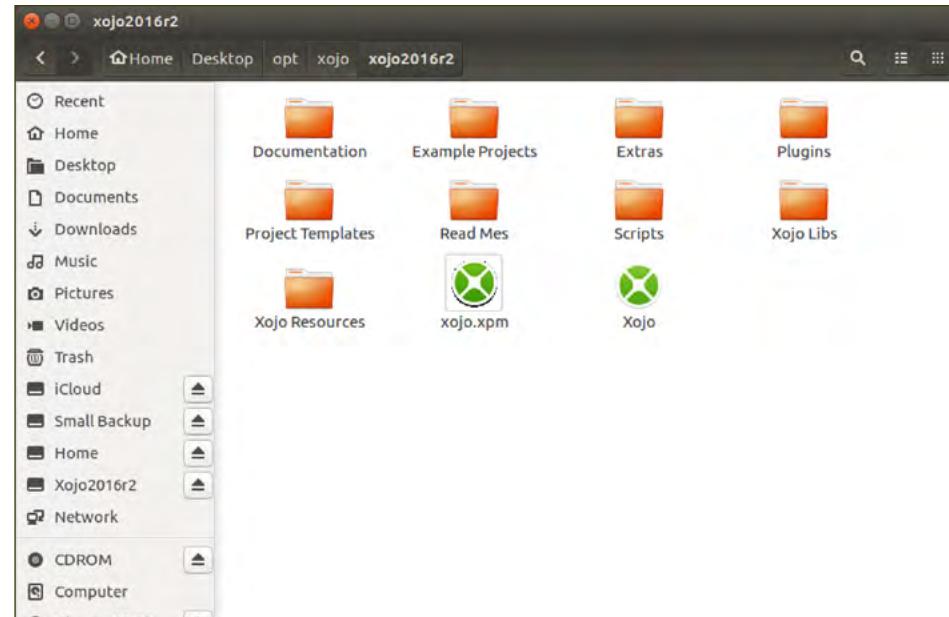


Windows: Doppelklicken Sie die Installationsdatei. Standardmäßig wird Xojo dann in den Programme-Ordner kopiert und eine Verknüpfung auf dem Desktop angelegt.





 **Linux:** Entpacken Sie die Archivdatei mit dem Archivmanager durch Doppelklick. Standardmäßig wird der Xojo-Ordner auf den Desktop kopiert. Sie können ihn an jeden beliebigen Ort verschieben.



 **Wichtig:** Obwohl sich mit Xojo durchweg 64 Bit-Programme erzeugen lassen, liegt die IDE selbst zurzeit (Stand 2016r2) nur als 32 Bit-Programm vor. Während dieser Umstand auf macOS und Windows keine besonderen Vorkehrungen benötigt, müssen Sie, falls Sie, was recht wahrscheinlich ist, eine 64 Bit-Installation einer Linux-Distribution benutzen, eventuell noch einige Bibliotheken nachinstallieren, damit Xojo läuft. Die aktuellen Informationen, welche das sind, finden Sie unter

<http://developer.xojo.com/system-requirements>



1.4. Die IDE



1.4.1. Der Project Chooser

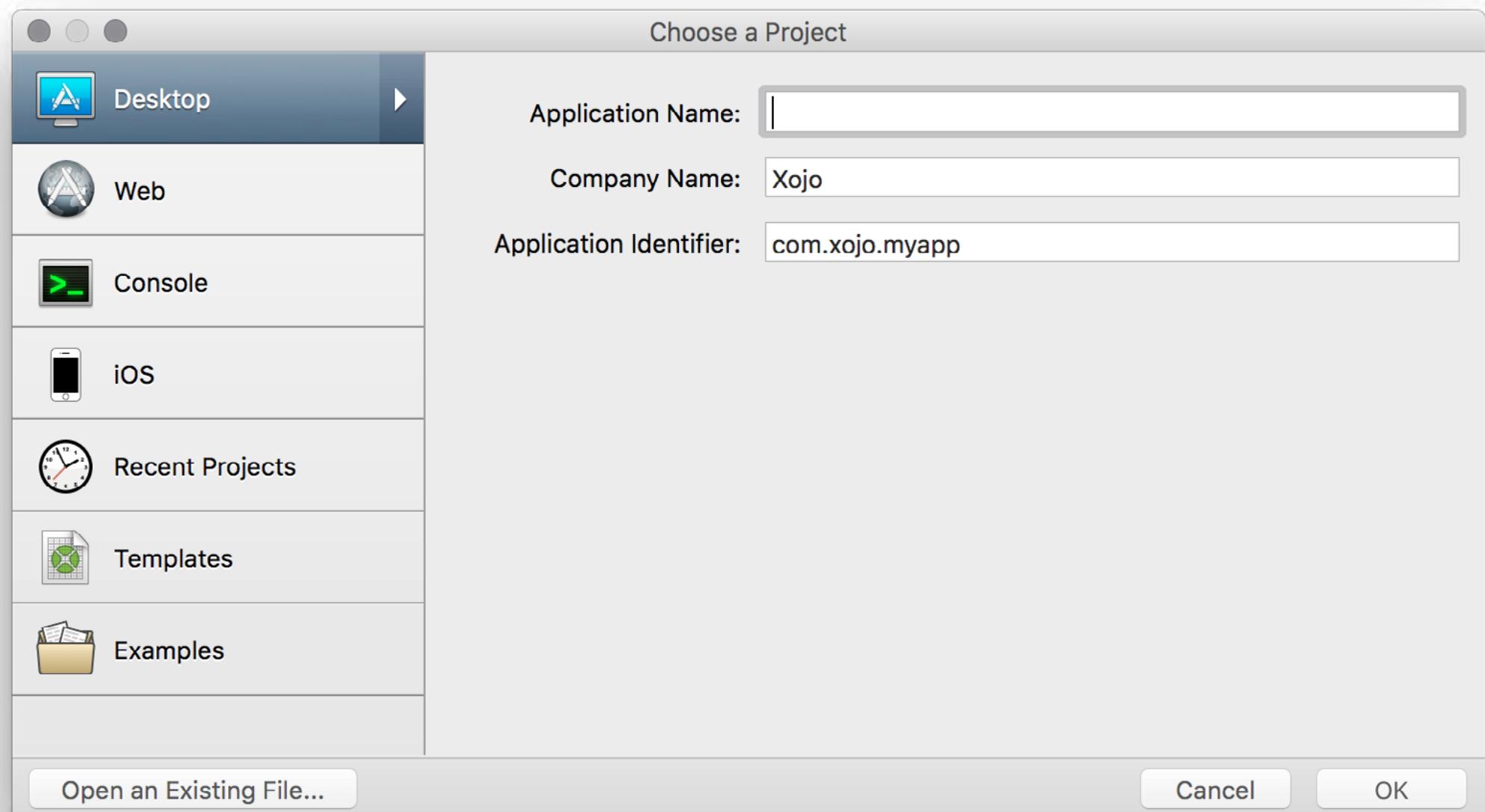
Wenn Sie jetzt Xojo per Doppelklick starten, wird sich das Bild zwar in Details unterscheiden, aber im Großen und Ganzen immer gleich aussehen. In den allermeisten Fällen werde ich mich darauf beschränken, die Mac-Version abzubilden. Einerseits ist das mein Lieblings-Entwicklungssystem, und andererseits ist hier auch die Retina-Unterstützung am weitesten vorangeschritten, soll heißen: Die Screenshots sind schärfer.

Nachdem Xojo nun also alle benötigten Bestandteile geladen und sie eventuell mit einer Lizenzmeldung begrüßt hat, landen Sie im Project Chooser.

Sie werden also zunächst aufgefordert, eine Projektauswahl zu treffen.

 **Die Bedeutungen** finden Sie, wenn Sie den Mauszeiger auf den Schaltflächen ruhen lassen.

 **Englische Terminologie:** Die Xojo-Programmiersprache ist dem Englischen entlehnt. Wie viele andere Programmierumgebungen auch liegt Xojo auch ausschließlich Englisch vor (mit der Ausnahme einer japanisch lokalisierten Fassung), was bedeutet: Sämtliche Menüs und Programmbestandteile tragen auch englische Namen. Wo es sinnvoll erscheint, werde ich auch eine Übersetzung liefern, ansonsten aber bei der Original-Benamsung bleiben, selbst wenn plausible deutsche Begrifflichkeiten existieren.





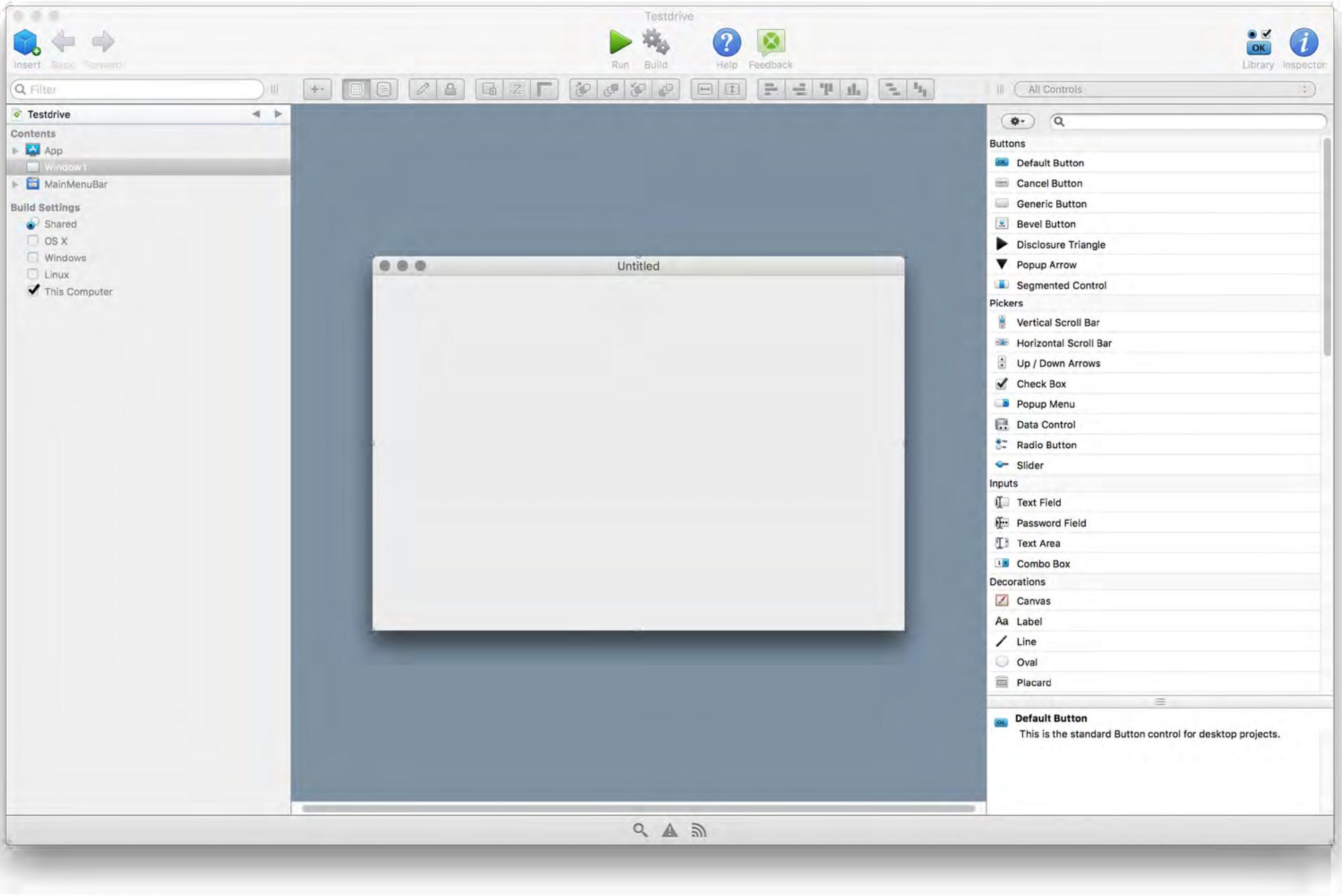
1.4.2. Das Hauptfenster

Jetzt sehen Sie, woher der Begriff integrierte Entwicklungsumgebung kommt: Dieses Fenster ist bis auf wenige Ausnahmen alles, was Sie bei der Arbeit mit Xojo sehen werden. Es vereint sämtliche wichtigen Programmierbestandteile unter einer Oberfläche. Was nicht heißen soll, dass man ganz ohne weitere Hilfsmittel auskommen sollte oder könnte: Ein individuelles App-Icon verlangt nach einem Grafikeditor, und die fortgeschrittene Arbeit mit Datenbanken etwa profitiert von einem vollausgewachsenen Datenbank-Editor. Aber das allermeiste der täglichen Routineprogrammieraufgaben wird hier vollumfänglich abgedeckt.

Hier ist das Hauptfenster eines jungfräulichen Desktop-Programms abgebildet. Je nach Zielplattform gibt es individuelle Unterschiede, auf die in den entsprechenden Plattform-Kapiteln noch in Länge und Breite eingegangen wird. Werden wird vielmehr ...

 **Erklärungen der Schaltflächen und Bereiche** finden Sie wieder, wenn Sie den Mauszeiger auf den Elementen ruhen lassen.

Die Funktionen im Detail werden in folgenden Kapiteln beleuchtet. Xojo tritt aber mit dem Anspruch an, ein RAD, ein Schnellentwicklungssystem, zu sein – soll heißen: Die Zeit zwischen Projektbeginn und erster Version ist äußerst niedrig. Lassen Sie uns diesen Claim einmal auf die Probe stellen; auch, um nicht zu viel in der Theorie zu gründeln ...



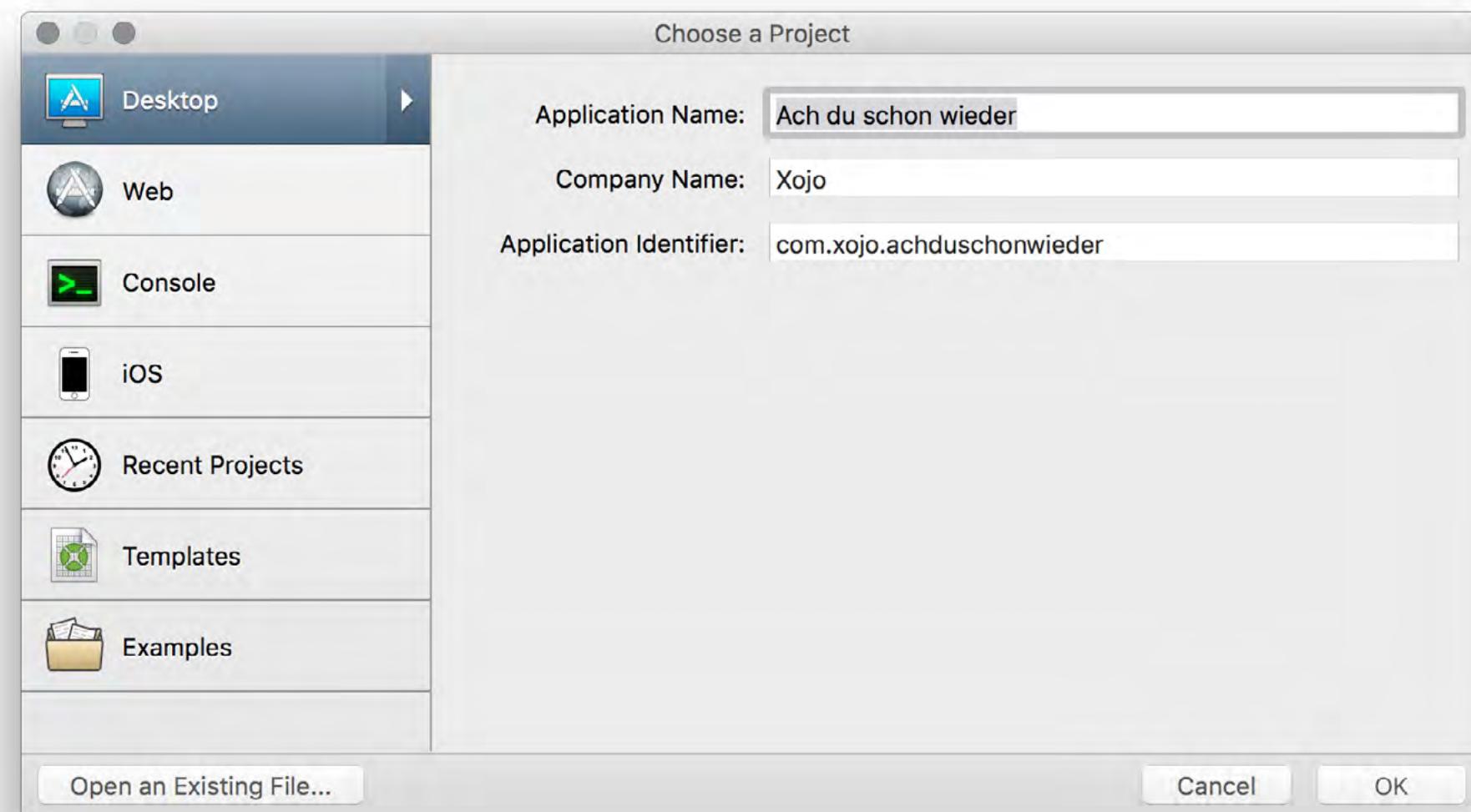


1.5. Das erste Programm

Ganz egal, welches Programmierhand- oder -nichthandbuch Sie aufschlagen: Als allererstes Programm finden Sie immer ein Äppchen, das in irgendeiner Form „Hallo Welt!“ auf den Bildschirm schreibt.

Aber wissen Sie was? In Jahren strenger empirischer Forschung habe ich herausgefunden, dass die Welt dadurch keine bessere wurde. Vielleicht ist sie sauer, weil nach dem Gruß nie etwas Konstruktives folgt. Setzen wir doch lieber Douglas Adams ein binäres Denkmal.

- ▶ Starten Sie Xojo und wählen Sie ein Desktop-Projekt.
- ▶ Geben Sie ihm einen hübschen Namen
- ▶ Klicken Sie auf „OK“.

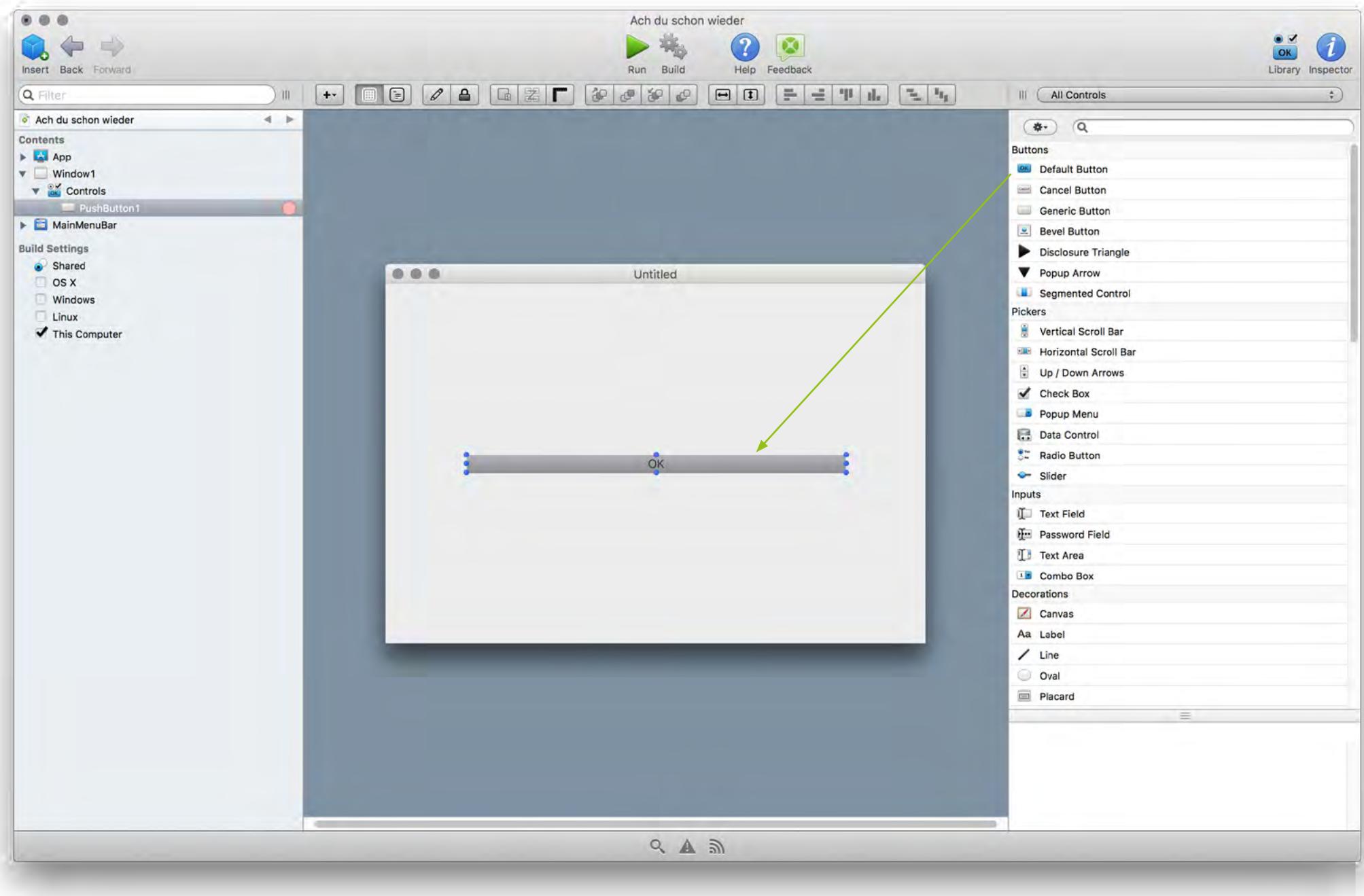




- ▶ Ziehen Sie den Default Button, das erste Objekt aus der Library, auf das Fenster im GUI-Editor.

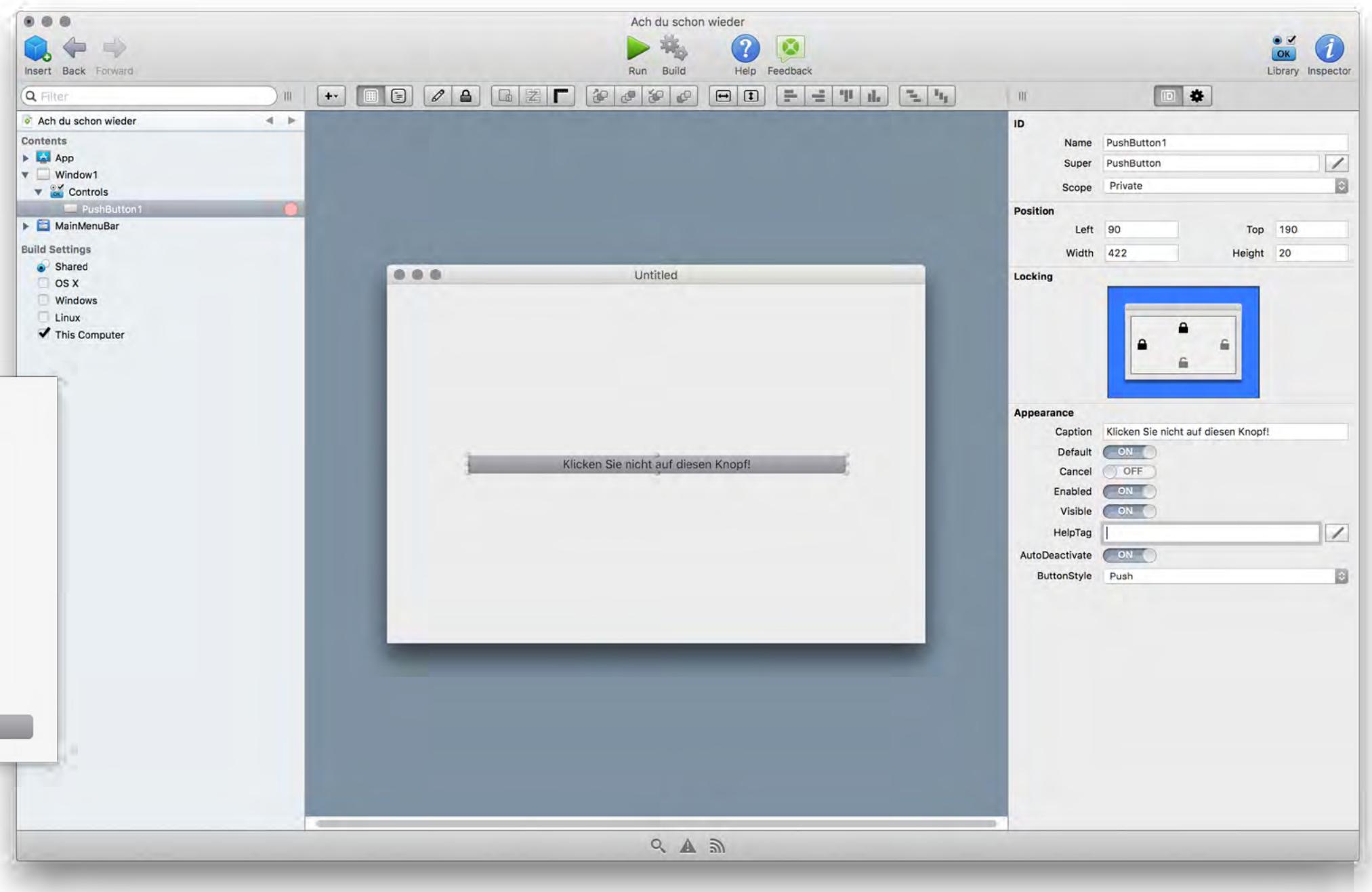
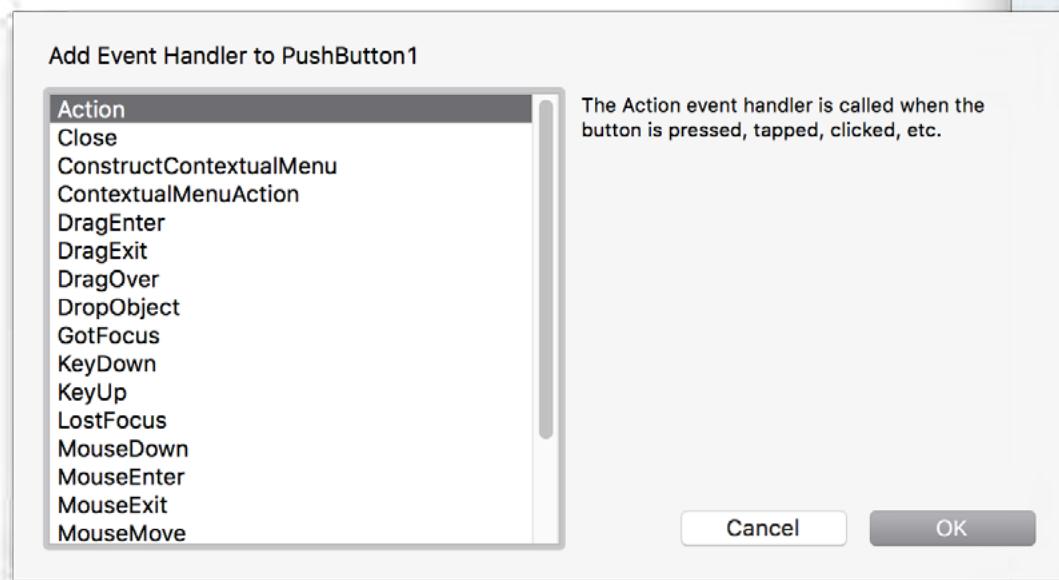
 **Wenn das rechte Panel keine Library anzeigt,** klicken Sie auf das Library-Symbol rechts oben in der Werkzeugleiste des Hauptfensters. Sollte die Liste anders aussehen als hier: Mit einem Klick in die Schaltfläche mit dem Werkzeugsymbol links am Kopf der Library können Sie die Darstellung ändern.

- ▶ Positionieren Sie ihn dort, wo es Ihnen gefällt. Das muss nicht die Mitte sein wie hier. Sie werden sehen, dass automatische Hilfslinien erscheinen, um Ihnen die Ausrichtung an den Fensterrändern und Standardabständen zu erleichtern.
- ▶ Ziehen Sie ihn etwas breiter, damit genügend Text hineinpasst.





- ▶ Klicken Sie nun auf das Inspector-Symbol rechts oben in der Werkzeugeiste des Hauptfensters.
- ▶ Geben Sie in das Inspector-Textfeld „Caption“ (was soviel wie „Bildüberschrift“ oder „Titel“ heißt) einen instruktiven Text ein.
Sie werden sehen, dass dieser Titel sozusagen als Live-Preview als Text des Buttons erscheint.
- ▶ Doppelklicken Sie auf den Button. Wählen Sie im erscheinenden „Add Action Event“-Dialog den ersten Eintrag, „Action“, und klicken Sie auf „OK“.



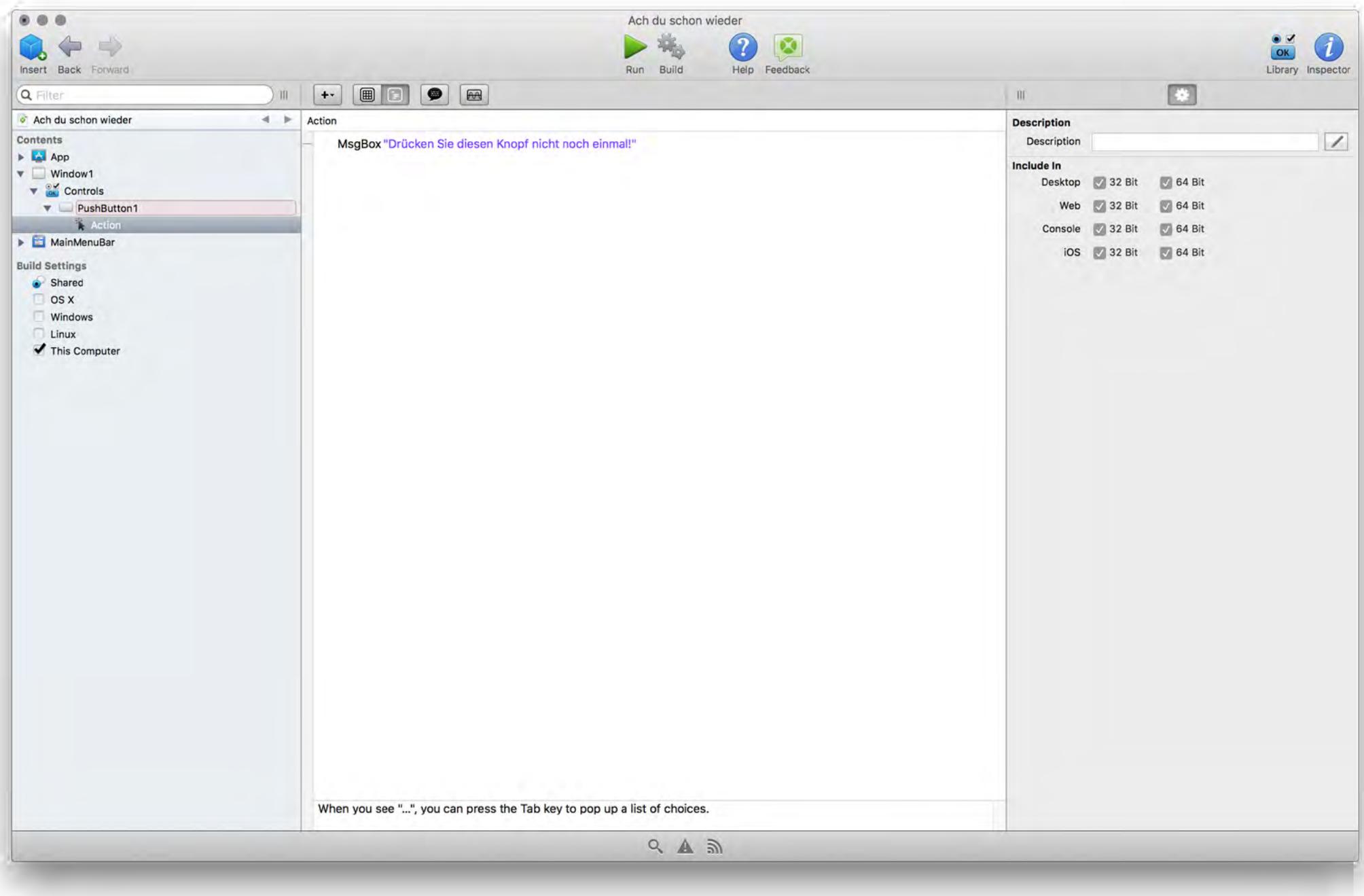


- Der Code-Editor erscheint, und anhand des Navigators sehen Sie, dass Sie nun den Code für den Action-Event des Buttons mit Namen „PushButton1“ bearbeiten (genauere Erklärungen auf den Folgeseiten). Geben Sie diese Zeile dort ein:

```
MessageBox "Drücken Sie diesen Knopf nicht noch →
einmal!"
```

- Drücken Sie jetzt auf das „Run“-Symbol in der Werkzeugleiste.
- Nach wenigen Sekunden erscheint das Programm auf Ihrem Bildschirm und fordert Sie auf, den Knopf nicht zu drücken. Das sollte dem Filmchen auf Seite 8 relativ ähnlich sehen.
- Wenn Sie genug mit dem Programm gespielt haben, beenden Sie es wieder, indem Sie in seiner Menüzeile den Menüpunkt „Programm beenden“ wählen.

 **Sollten Sie eine Fehlermeldung erhalten,** kontrollieren Sie alle Eingaben anhand der Abbildungen genau. Sollte es immer noch nicht funktionieren, laden Sie das Demo-Programm aus dem Anhang. Nach Möglichkeit sollten Sie aber immer versuchen, Ihre Anwendungen selber zum Laufen zu bewegen, um Ihren Lieblings-Programmierfehlern auf die Spur zu kommen. Im Programmierbereich ist die alte Binsenweisheit „Aus Fehlern lernt man“ so wahr wie selten sonst.





1.5.1. Ging schnell, hat funktioniert – was sagt mir das?

Zunächst einmal sollten Sie nun eine Aussage treffen können, ob Xojo in der Tat als RAD qualifiziert.

Die kleine Demo mag ein äußerst fipsiges Programm mit sehr spezialisierter Funktionalität gewesen sein, das Sie jetzt vielleicht als gar nicht vollwertig betrachten. Aber ist Ihnen aufgefallen, dass es sich hinsichtlich Fensterverhalten und Menüs wie jedes andere vollausgewachsene Programm benommen hat?

Es steckt also viel mehr Intelligenz darin, als Sie bewusst hineinprogrammiert haben, und das bringt uns wieder zur Kochrezeptmetapher von Seite 7:

Darin wäre Xojo die vollausgestattete, automatisierte Küche, die Ihnen schon die notwendigen Grundzutaten für die gewünschte Speiseart bereitstellt.

Der Button wäre ein solches elektronisches Küchengerät, und wir haben ihm gesagt, was er tun soll, wenn sein Action-Event eintritt.

Ein

Event,

zu Deutsch Ereignis, ist in Xojo streng genommen nichts anderes als ein Platzhalter für Programmcode, der automatisch dann aufgerufen wird, wenn ebenjenes Ereignis eintritt.

Der

Action-Event

eines Buttons ist dabei, das hat Ihnen auch der erklärende Text im „Add Action Event“-Dialog erläutert, der Code-Platzhalter, der dann aufgerufen wird, wenn ein Button gedrückt wird.

Ein

Button bzw. PushButton

bedarf wenig Erläuterung: Das ist eines der am häufigst benutzten Steuerelemente in Desktop-Anwendungen und besteht aus einer beschriftbaren Schaltfläche (wobei die Fläche wie aktuell in iOS auch unsichtbar sein kann), die auf Berührung resp. Klick reagiert. Der

DefaultButton

ist einfach eine Abart davon, nämlich der Button, der als Standardvorgabe (= Default) definiert wurde und deshalb hervorgehoben angezeigt und durch die Eingabe von Return aktiviert wird.

Der Befehl

MsgBox

schließlich bringt eine Dialogbox auf den Bildschirm, ein Fenster mit sehr eingeschränkten Gestaltungsmöglichkeiten, das in der Standardvariante einen Text anzeigt und auf die Bestätigung seines „OK“-Buttons wartet.

So, wie wir in der einführenden Kochbuchmetapher dem Backofen Anweisungen gaben, uns zu informieren, wenn er seine Flugtemperatur erreicht hat, haben wir durch Füllen des Button-Action-Events dem Button gesagt, was er tun soll, wenn der Benutzer auf ihn klickt.

Also haben Sie damit nicht nur das erste Xojo-Programm geschrieben, sondern ggf. auch Ihr erstes objektorientiertes Programm. Ist gar nicht so schwer, oder?

Und was die Größe angeht: Als Mensch um die 1,70 bin ich ohnehin Unterstützer der „Size doesn't matter“-Fraktion. Weitau mehr aber ist es auch so, dass eine Vielzahl von Programmen gar nicht so viel mehr benötigt als ein paar Zeilen Code, um als respektiertes Mitglied der App-Welt zu gelten. Die Abfrage eines Internetdienstes etwa benötigt nur wenige Zeilen, die zugehörige Visualisierung oftmals auch. Einen eigenen Webbrowser, zumindest mit Grundfunktionalität, erstellen Sie in Xojo in wenigen Minuten. Undundund ... also kein Grund zur falschen Bescheidenheit!

Die etwas größeren Programme folgen eh in Kürze – lassen Sie uns mal weiterschauen!



1.6. **If Interesse = ...**

Entscheiden Sie, wie die Fahrt weitergeht!

Hier ein Stückchen Phantasiecode, der Ihnen bei der Entscheidung helfen soll. Geben Sie ihn nicht in Xojo ein – er ist für Ihren eigenen Prozessor geschrieben!

```
If Interesse = "Programmieren lernen" Then
    "Einführung in die Programmierung".Lesen
ElseIf Interesse = "Objektorientiertes Programmieren lernen" Then
    OopOhWeh.Lesen
ElseIf Interesse = "Praktisch lernen" Then
    QuickStartTutorials.Lesen
    IntroVideos.Anschauen
Elseif Interesse = "Entwicklungssystemumstieg" Then
    Dim KapitelNummer As Integer = Xojo.Math.Randomint(5, MaxKapitel)
    Dim ZufallsKapitel As BuchKapitel = Nichthandbuch.Kapitel(KapitelNummer)
    ZufallsKapitel.Lesen
End If
```



EINFÜHRUNG IN DIE PROGRAMMIERUNG

Ein Einsteigerkapitel, das sich insbesonders anbietet, wenn Sie noch niemals vorher programmiert haben – oder es versucht, aber so recht nicht verstanden. Sie finden hier:

- Grundbegriffe der Programmierung: Bits und binäre Zahlen, Variablen, Schleifen, Bedingungen, Methoden, Bugs
- Datentypen: Integer, String, Text, Single & Double, Boolean
- Operatoren: +, -, ., /, Mod, <, >, = etc.
- Beispielprojekte: Kommandozeile



KAP. 2: Einführung in die Programmierung

2.1. Schriftliche Bekanntmachung der Tagesordnung



Hmpf? Naja, zumindest sind das zwei Definitionen der ursprünglichen Bedeutung des griechischen *prόgramma*, und wenn man sich mit neuen Themen auseinandersetzt, hilft ein Blick in die Wortherkunft häufig beim Verständnisverlust.

Wir sind in der Praxis gar nicht mal so weit weggekommen von den Ursprüngen: Auch im Computersinn wird ein Programm schriftlich erstellt, und es legt die Ablaufordnung fest. Aber die Ablaufordnung für was?

Hier der sehr vergrößerte Blick in einen Computer. Wir haben eine Kombination von Ein-Ausgabegeräten auf der einen Seite (Monitor, Tastatur, Maus etc.), eine Ansammlung von Speicher auf der anderen und dazwischen eine CPU, die die Verarbeitung von Benutzereingaben und daraufhin des Arbeitsspeichers und die Vermittlung bzw. Übersetzung zwischen Speicher und Anwender erledigt.

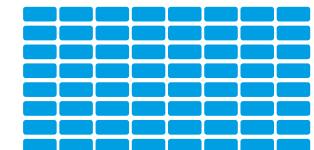
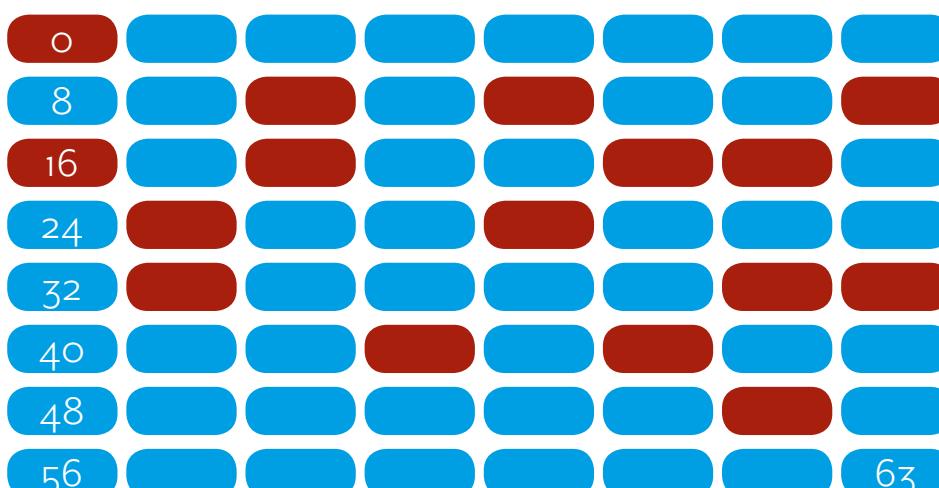


Abb. 3: Viel mehr braucht der Computer nicht: Ein-/Ausgabegerät, Prozessor, Speicher

Nehmen wir den Speicher unter die Lupe, dann sehen wir, warum behauptet wird, herkömmliche Computer besäßen eine

Binäre Logik:



Der Speicher besteht eigentlich nur aus einer großen Ansammlung von Speicherstellen, den

Bits,

die genau wie ein Schalter nur zwei verschiedene Zustände annehmen können: an oder aus, in Zahlenwerten 0 oder 1. Hier im Beispiel soll Rot = an und Blau (damit es nicht so trist wird mal kein Schwarz) = aus bedeuten.

Ach, übrigens: Obwohl es nach einem Bisschen klingt, ist Bit ein Kunstwort für **Binary Digit**: Binäre Zahl.

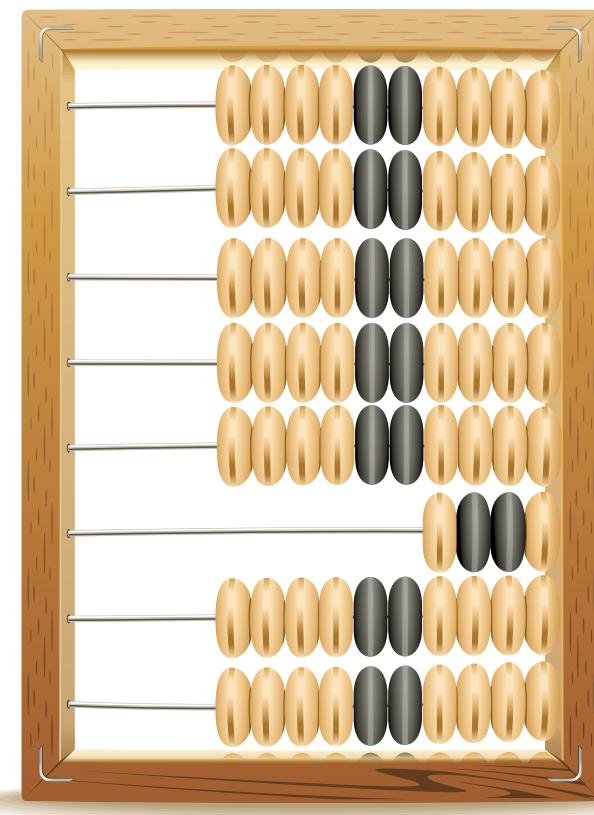
In nebenstehender Illustration sind die Bits in Reihenfolge durchnummiert. Im Gegensatz zum üblichen Sprachgebrauch (bis auf wenige Ausnahmen gilt dabei):

Der Computer beginnt die Zählung immer bei 0.

Das mag sich jetzt nach trivialer Information anhören, aber machen Sie an dieser Stelle besser ein Eselsohr ins PDF: Diesen Umstand zu vergessen ist eine der häufigsten Fehlerursachen. **Wenn Sie sich wie in der Abbildung 64 Bit reservieren lassen, dann haben diese die Nummern 0 bis 63, nicht 1 bis 64!**

Und wie rechnet der Computer jetzt damit? Im kleinen Zahlenaum wäre ja noch denkbar, das Bit mit der entsprechenden Nummer zu nutzen, aber wenn man mit Millionen oder größeren Beträgen jongliert, wäre selbst ein gut mit Arbeitsspeicher ausgestatteter Rechner bald am Ende seiner Kapazität.

Überraschung: Er macht es genau wie wir, zumindest im Schriftlichen oder mit dem Abakus. Erinnern Sie sich noch daran? Verschiedene Reihen von Schiebeelementen stellen die Zehnerstellen einer Zahl dar: Von Einern über Zehner, Hunderter, Tausender usw. kann man mit einem kleinen Abakus schon eine ganz beachtlich große Zahl darstellen.



Einer
Zehner
Hunderter
Tausender
Zehntausender

Abb. 4: Strenggenommen kein Abakus, sondern eine russische Stschoty, bei der einige Kugeln andere Wertigkeiten besitzen. War aber das schönste Bild, das ich finden konnte – ignorieren Sie die abweichenden Reihen & Farben!

Will man also eine 1 darstellen, bewegt man die erste Kugel der Einer-Reihe auf die andere Seite, für eine 10 die erste Kugel der Zehner-Reihe, und eine 11 ergibt sich durch Bewegen der ersten Kugeln beider Reihen.

Das sich diese Zahlenreihen zur Basis 10 ergeben, liegt in der Natur der Sache – also des Menschen. In der Standardausstattung wird dieser mit zehn Fingern geliefert, hat also die erste Abakus-Reihe schon als natives Feature on board.

Mathematisch korrekter können wir sagen, dass die Abakus-Reihen Zehnerpotenzen in nach unten steigender Wertigkeit abbilden.

Potenzen

– erinnern Sie sich noch? – sind definiert als die Multiplikation einer Zahl mit sich selbst. Dabei wurde die 0te Potenz als 1 festgelegt. Statt der hübschen Begriffe in der Illustration links lässt sich also auch schreiben:

$$10^0 = 1$$

$$10^1 = 10$$

$$10^2 = 100$$

$$10^3 = 1000$$



$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \\ 2^5 &= 32 \\ 2^6 &= 64 \\ 2^7 &= 128 \end{aligned}$$

Abb. 5: Der Computer-Abakus

Der Computer besitzt keine 10 Finger (in Nummerierung 0–9), sondern nur 2: 0 und 1. Dafür reicht dann eine Kugel pro Zweierpotenz-Reihe: Welchen Wert die Reihe dann hat, ergibt sich aus der Position der Kugel. Im obigen Beispiel wäre also der Zahlenwert 0 dargestellt.

Bewege ich die Kugeln in den Reihen 0, 3 und 7 nach links, ergibt sich

$$1 \cdot 2^0 = 1 \cdot 1 = 1$$

$$1 \cdot 2^3 = 1 \cdot 8 = 8$$

$$1 \cdot 2^7 = 1 \cdot 128 = 128$$

$$\begin{array}{r} \text{Summe} \\ \hline 137 \end{array}$$

Summe 532

Und wie geht der Computer jetzt an diese Aufgabe?

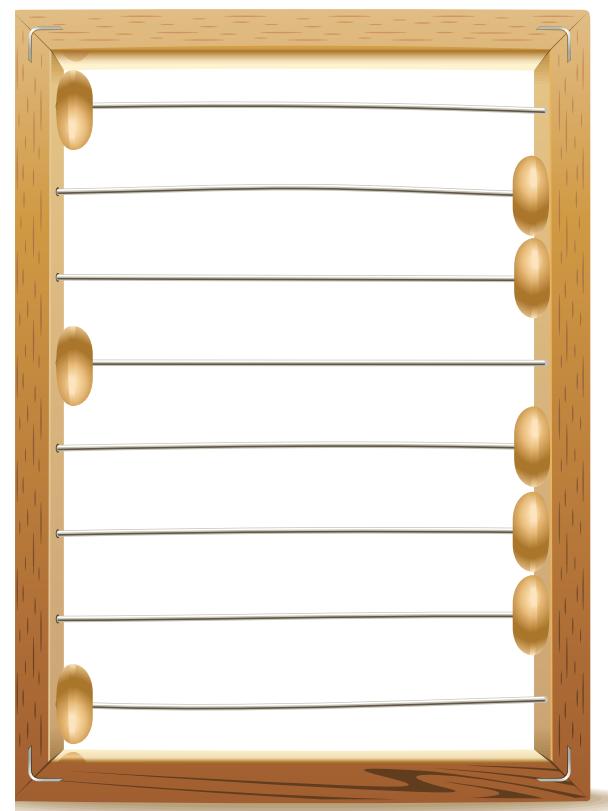


Abb. 6: 137!

Illustriert sieht das so aus. Sie sehen: Das ist alles keine Raketentechnik, sondern Grundschulmatematik. Wenn wir jetzt den Abakus durch unsere Speicherbits ersetzen und alles um 90° nach rechts drehen, landen wir hierbei:

$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

Sollte dies alles neu für Sie gewesen sein, gehen Sie an dieser Stelle nicht zu schnell weiter! Malen Sie ein paar Binärzahlen auf einen Zettel, vergleichen Sie sie ggf. mit einem Taschenrechner (viele, auch der in macOS integrierte, besitzen direkte Binärdarstellungen und -Umrechnungen), bis Sie sicher sind, dass das Konzept sitzt!

Diese logische Zusammenfassung von 8 Bit zu einem Gesamt-wert nennt sich

Byte.

Sozusagen ein Haps. Auch wenn mitunter einzelne Bits verwaltet werden: In der Praxis stellt es die kleinste Verwaltungseinheit dar, die ein Computer zu beackern pflegt.

Was das angeht: Wie steht es um den Zahlenraum, den das Byte darstellen kann? Minimal liegt dieser bei $0 =$ alle Bits aus, das ist klar. Sind alle Bits an, ergibt sich

$$\begin{aligned} 2^0 &= 1 + 2^1 = 2 + 2^2 = 4 + 2^3 = 8 + 2^4 = 16 + 2^5 = 32 + 2^6 = 64 + 2^7 \\ &= 128 = 255 \end{aligned}$$

Ein Byte kann den Zahlenraum vom 0 bis 255, also 256 verschiedene Werte annehmen.

Und wenn man größere Zahlen benötigt?

Dann nimmt man nicht ein Byte, sondern mehrere von ihnen, deren Bitwertigkeiten dann einfach weiter durchnummieriert werden. Mit zwei Byte ergeben sich dann (das rechne ich Ihnen jetzt nicht vor – erweitern Sie die obige Rechnung einfach bis 2^{15}): 65.536 verschiedene Werte, also 0–65.535.

Da jedes neue Bit zu einer Verdoppelung der „Kapazität“ führt, reichen wenige Bytes zur Darstellung enorm großer Zahlenwerte.

2.1.1.

Und wenn ich mehr als ganze Zahlen will?

Sehr gute Frage! Sie haben völlig Recht: Mit einem Computer lassen sich nicht nur Ganzzahlen verwalten. Manchmal spuckt er auch Text aus (oder scheint ihn zu verstehen), und mit Zahlen, die auch hinter dem Komma noch etwas zu vermelden haben, sowie mit negativen Werten scheint er ja auch gut klarzukommen.

Wie war das noch mal mit dem Kochrezeptgleichnis? In hemmungsloser Nostalgie behauptete ich, die Dinger würden stets mit einem „Man nehme ...“ beginnen. Und dann greift der Kochfeldoperator¹ nach Schüssel, Salz, Pfanne oder Biotonne, je nachdem, was vorgegeben ist.

Das funktioniert alles, weil sich die Dinge unterscheiden und wir auf den ersten Blick (meist zumindest) Brot und Brotmesser zu identifizieren in der Lage sind.

Der Computer hat's da schwerer. Er besitzt nur seine Bits, die er zwar zu größeren Byte-Strukturen zusammenzählen kann, doch damit hat es sich. Bits gibt es nicht in verschiedenen Geschmacksrichtungen und Farben, für Ganzzahlen, Fließkom-mazahlen, Texte und Wasweißlich.

Falls Sie einmal Kind waren oder selbst in den Besitz eines solchen gelangt sind, kennen Sie die Antwort: „Das Waschbecken ist jetzt mal die See und die Seifendose ist ein Dampfer!“

¹ Bisher konnte ich's umschiffen, nun aber der Hinweis: Sie finden hier keinerlei gegenderte Sprache. Es geht mir um die Vermittlung von Information, nicht um sprachlich/politische Korrektheit, die auf einer grandiosen Missinterpretation beruht und den eigentlichen Sinn von Sprache erschwert. Mehr sag ich nicht dazu.



Wir legen zusammen mit dem Computer einfach fest, welche Bedeutung ein Byte – oder mehrere davon, zu einem Verwaltungsblock zusammengezählt – erhalten soll.

In Xojo dient dazu der Begriff

Dim ... As

und man kann ihn sich ganz gut mit „Dimensioniere“ merken – also soviel wie Gib mir ein (paar) Byte, das/die eine Ganzzahl darstellen soll(en)“. Dazu gehört immer ein **As**, also „als“, das den Datentyp definiert. In Xojo liest sich das dann wie

```
Dim MeineZahl As Integer
```

Hier einmal farbig so hervorgehoben, wie Xojos Code-Editor in der Standardeinstellung das auch zu tun pflegt – im Folgenden mache ich das ggf. nur bei neu vorgestellten Features. Er hebt reservierte Befehlsworte hervor (und sorgt auch für automatische Einrückungen, wo sie der Struktur dienlich sind) – damit sind irrtümlich als Variable benutzte Kommandobegriffe z. B. schnell auszumachen, und ein Vertipper kann flink eliminiert werden.

MeineZahl also ist eine

Variable

Wir können den Speicherbereich, in dem wir jetzt diese Ganzzahl speichern und manipulieren können, unter seinem Variablenamen ansprechen. Eine Zeile wie

```
MeineZahl = 100
```

weist dieser Variablen den Wert 100 zu.

Eine Variable vom Datentyp

Integer

kann nur Ganzzahlen aufnehmen, also keine Komma-Werte.

Wieviele Bits ein Integer-Wert belegt und wie groß der Datenbereich ist, den ich damit abdecken kann, hängt vom verwendeten Betriebssystem und von der Integer-„Geschmacksrichtung“ ab, die ich bei der Deklaration – dem Erklären des Speicherbereichs **MeineZahl** – verwende:

Auf einem 32 Bit-System ist ein Integer 32 Bit groß, auf einem 64 Bit-System 64 Bit.

Das ist vielleicht nicht immer, was ich will. Für individuelle Größendefinitionen unabhängig vom verwendeten Betriebssystem bietet Xojo die Untertypen

UInt8, UInt16, UInt32, UInt64

Wie nicht schwer zu erraten, geben die Zahlen die Bitgröße des Integers an. Das „U“ steht für unsigned, unsigniert: Es sind stets positive Zahlen!

Um das Beispiel der Vorseite in Xojo zu erläutern: Den Computer-Abakus mit Wert 137 simuliere ich durch

```
Dim Abakus As UInt8
Abakus = 137
```

oder kurz durch

```
Dim Abakus As UInt8 = 137
```

Will man nun gar keinen Ganzahlwert verwalten, sondern ein Stückchen Text – sagen wir mal, ein sehr kleines, nämlich nur ein Zeichen –, dann muss eine Variable mit einem entsprechenden

Datentyp

deklariert werden. Ein Datentyp definiert die Art, wie der Computer einen bestimmten Speicherbereich interpretiert. Traditionell dient dazu der Datentyp

String

dessen beste Übersetzung wohl „Zeichenkette“ ist. Wollen Sie einem String einen Wert zuweisen, gehört dieser zwischen zwei Gänsefüßchen. Entsprechend erzeugt der Code

```
Dim MeinZeichen As String = "A"
```

oder, wenn Sie es expliziter mögen

```
Dim MeinZeichen As String
MeinZeichen = "A"
```

die Reservierung eines Speicherbereichs, der, wenn wir ihn unter die Lupe nehmen, so ausschaut:

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Die Codierung einfacher Zeichen nach der [ASCII](#)-Norm sieht nämlich vor, dass der Zeichenwert A dem Integerwert 65 entspricht. Sie erinnern sich: „Die Badewanne ist jetzt mal ...“ – der Computer kennt nur seine Bits. Wie er sie interpretiert, ist von unserer Abmachung mit ihm abhängig.



 **Bitte beachten Sie:** Das ist bis jetzt nur die halbe Wahrheit zum Thema Integer, und den Datentyp String werden wir zwar noch genauer anschauen, ihn dann aber schnell ad acta legen. Er ist mittlerweile durch einen moderneren Datentyp ersetzt worden, allerdings noch nicht vollumfänglich.

Bevor wir nun endlich ein wenig in die Praxis gehen, noch schnell ein dritter Datentyp, der implizit schon mehrfach angesprochen wurde, bzw. eigentlich zwei davon:

Single und Double

sind Fließkommadatentypen, d.h., wir können endlich auch Zahlen verarbeiten, die nach dem Komma noch Stellen aufzuweisen haben.

Wenn Sie sich die Binärstruktur des Computers noch einmal anschauen, werden Sie feststellen: Für Kommazahlen ist da eigentlich nicht recht Platz. Es gibt keine Vor- und Nachkommabits. So sind es denn auch recht komplexe mathematische Operationen, die bei Fließkommaberechnungen herangezogen werden, und sie belasten die CPU auch weitaus stärker als Integer-Funktionen, die ja direkt in Bits abzubilden sind.

Weshalb man bei den beiden Fließkommatypen, die sich in Ihrer Bitgröße unterscheiden (Single benutzt 4 Bit, Double 8) auch weniger vom darstellbaren Zahlenraum als von der Genauigkeit sprechen kann, mit dem in diesem gerechnet wird.

Und auch bei Double-Genauigkeit gilt:

Fließkommawerte sind oft nur Näherungswerte:
Der Computer ist kein Taschenrechner!

Sie kennen das Phänomen eventuell von Layout- oder Grafikprogrammen, wenn Sie einen ganz sauberen Wert eingeben, das Programm aber steif und fest behauptet, dass ihm ein paar Kommastellen besser zu Gesicht stünden. Dann hat ein Rundungsfehler zugeschlagen.

 In der Praxis scheint sich die Performance nichts zu nehmen: Es ist kein eindeutiger Geschwindigkeitssieger festzustellen. **Auf einen Single-Datentypen zu gehen, um Rechenzeit zu sparen, scheint mir keine erfolgversprechende Angelegenheit.** Gönnen Sie sich ruhig die höhere Double-Genauigkeit, es sei denn, die Funktion, mit der sie arbeiten wollen, versteht sich ohnehin nur auf Singles.

2.1.2. Negative Stimmung?

Auf der Vorseite habe ich die Integer-Untertypen UInt8 und Konsorten kurz vorgestellt. Die reine Integer-Entsprechung gibt es auch, nämlich

UInteger

– ein unsignierter Integer, der 32 Bit auf 32 Bit-Systemen und 64 Bit auf 64 Bit-Systemen belegt.

Wie erwähnt steht das „U“ für Unsigned, unsigned, also nicht mit einem Vorzeichen behaftet – bzw. auch nicht mit einem solchen ausstattbar. Kurz gesagt werden bei allen U-Integers sämtliche Bits zur Darstellung einer positiven Zahl verwendet. Entsprechend gilt auch das Computer-Abakus-Modell bei den U-Integers vollumfänglich: Ein UInt8 kann Werte von 0 bis 255 darstellen usw.

Manchmal braucht man Zahlen aber auch im negativen Bereich. Weshalb reine Integers, also solche ohne U im Namen, einen eingeschränkteren Datenbereich haben. Man benutzt nämlich ihr linkstes Bit als Vorzeichen: Ist es gesetzt, wird die Zahl als negativ betrachtet; ist es leer, haben wir einen positiven Integer vor uns. Dies hier wären also nach wie vor **127**:

neg. 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Während das gesetzte Negativ-Bit im Int8 dann keine 255, sondern eine **-128** aus der ganzen Sache macht:

neg. 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Da ja eine -0 wenig Sinn ergibt, gönnt man sich den Luxus eines leicht verschobenen Datenbereichs. Wie die Bits beim gesetzten Negativ-Bit zu interpretieren sind, darüber existiert keine Norm – es gibt unterschiedliche [Modelle](#). Ich gehe daher nicht im Detail darauf ein und liefere Ihnen stattdessen eine Übersicht über die

Integer-Datenbereiche

Name	Größe	Min.	Max.
Int8	1 Byte	-128	+127
UInt8	1 Byte	0	+255
Int16	2 Byte	-32.768	+32.767
UInt16	2 Byte	0	+65.535
Int32	4 Byte	-2.147.483.648	+2.147.483.647
UInt32	4 Byte	0	+4.294.967.295
Int64	8 Byte	-9.223.372.036.854.775.808	-9.223.372.036.854.775.808
UInt64	8 Byte	0	+18.446.744.073.709.551.615



2.2. Nun aber zur Tagesordnung!



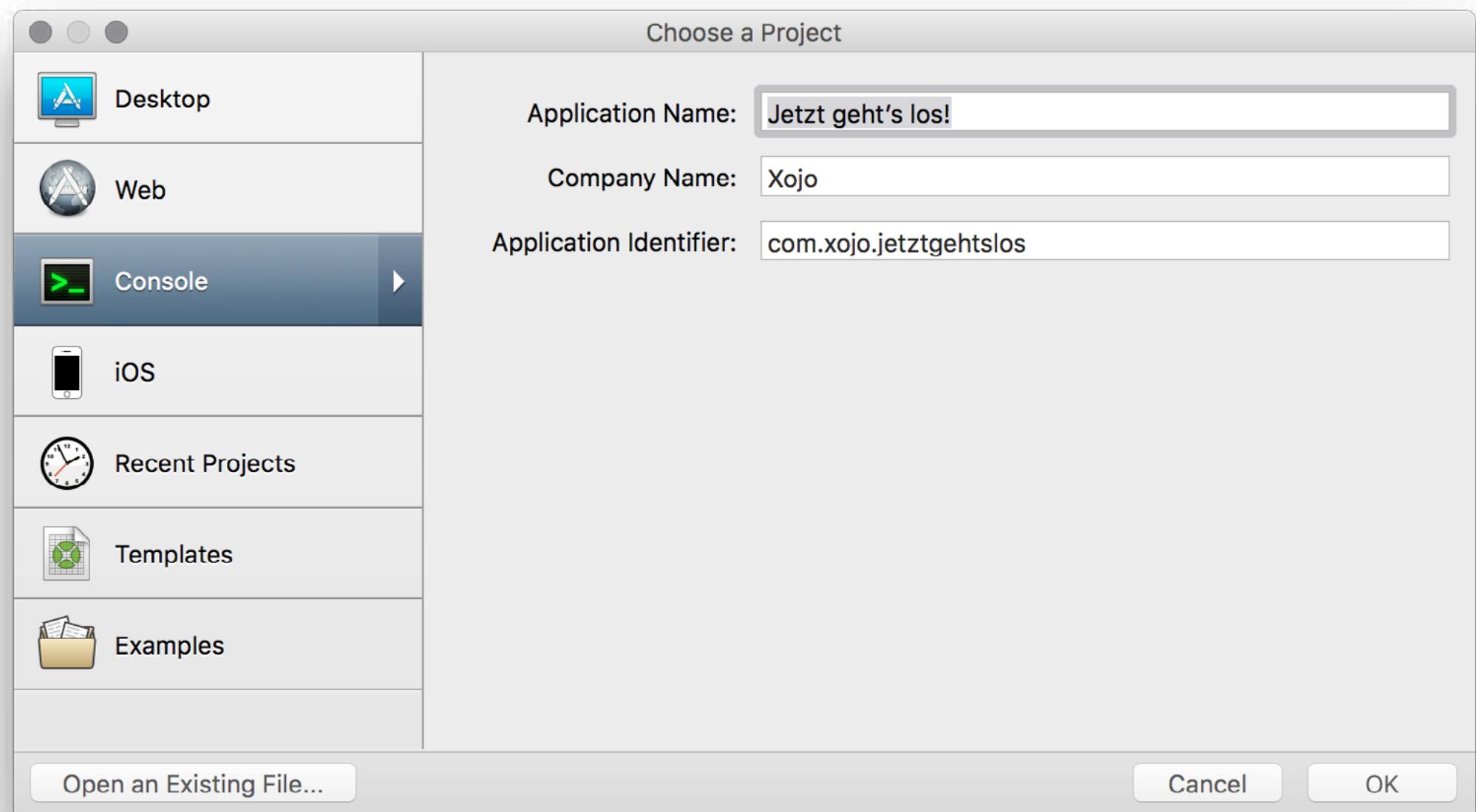
Endlich mal ein wenig Praxis (und vielen Dank fürs Durchbeißen bis hier)!

- ▶ Starten Sie Xojo und wählen Sie im Project Chooser ein neues Console-Project.

 **Wie – Kommandozeile?** Ja. Die Kommandozeile bietet eine Umgebung, die dem linearen Programmieren sehr nahekommt, und wir können uns die Objektorientiertheit als nächsten Schritt vorbehalten.

- ▶ Geben Sie dem Projekt einen wohlklingenden Namen.
- ▶ Klicken Sie auf OK.

 Falls Ihnen **Console** oder **Kommadozeile** nichts sagen sollte: In der Regel wird man auf den modernen Desktop-Betriebssystemen selten genötigt, per Text-Befehlseingabe mit System- oder systemnahen Programmen sprechen zu müssen. Hin und wieder passiert dies aber schon mal und fühlt sich durchaus etwas steinzeitlich an. In der Praxis sind viele Systemtools, die über hübsche Oberflächen verfügen, oft nichts anderes als angenehmer zu bedienende grafische Schaltflächen für die eigentlichen Systemprogramme, die ihrerseits auf optisch getunete Ausgaben verzichten.





Sie sehen jetzt das Hauptfenster für ein Kommandozeilenprojekt vor sich. Die Library ist viel übersichtlicher als bei Desktop-Projekten. Was kein Wunder ist: Wir haben hier keine grafischen Objekte wie Fenster, Buttons und dergleichen zur Verfügung.

Entsprechend ist der Editor-Bereich leer, und der Navigator zeigt ein einziges Objekt: Das

App-Objekt,

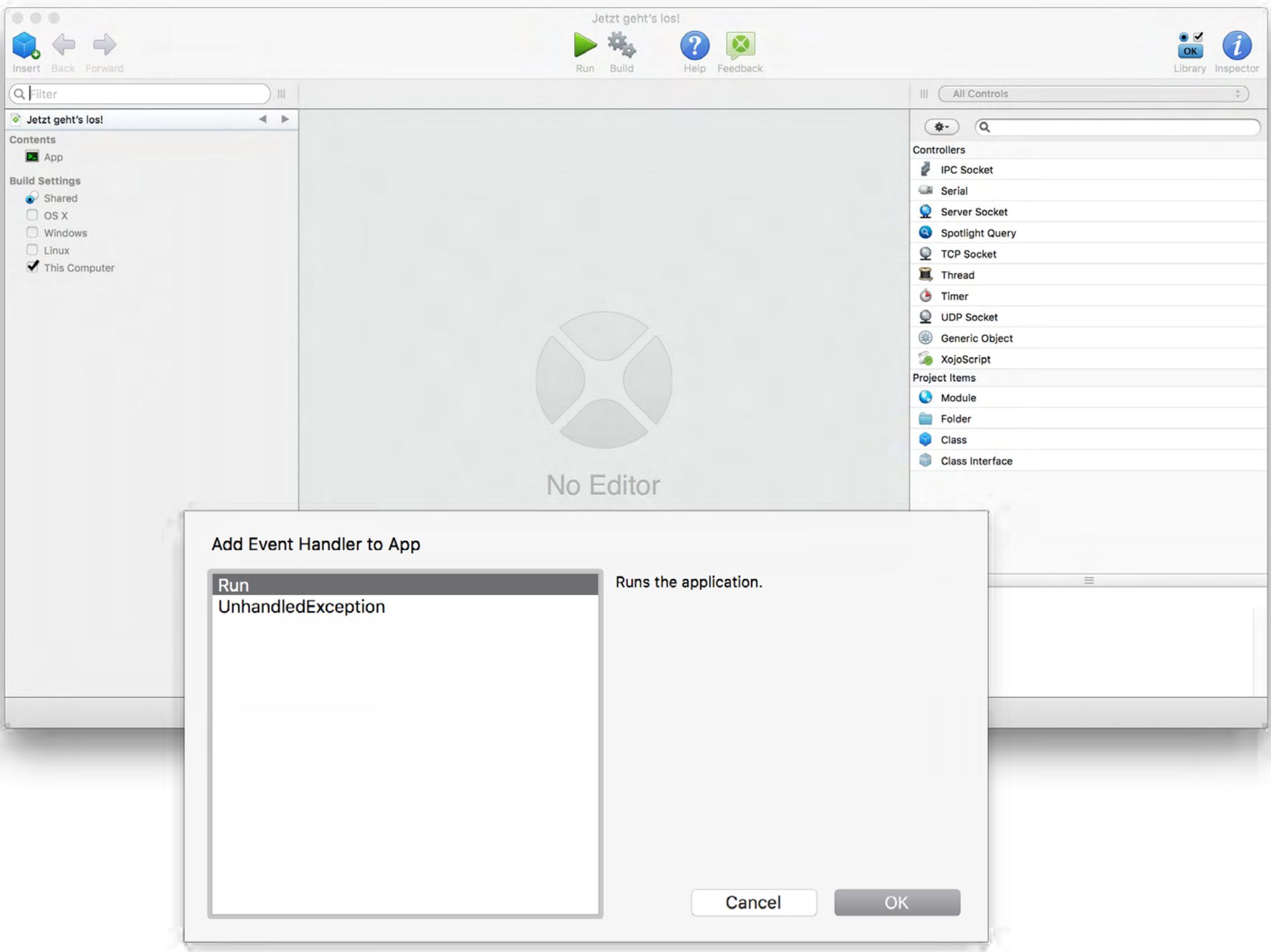
stellvertretend für die Anwendung selbst.

Diese Anwendung wird beim Programmstart geöffnet, und so, wie ein jegliches grafisches Objekt dann einen Open-Event feuert, besitzt auch das App-Objekt einen Event, den es dann aufruft. In diesem Fall heißt er „Run“.

Feuern

wäre eine weitere Vokabel für Ihr Heftchen: Wie auf Seite 21 beschrieben, ist ein Event ein Platzhalter für Programmcode. Ein Event schließt daher ins Blaue, und steht zufällig Code an der richtigen Stelle – existiert ein Eventhandler –, dann trifft es ihn. So zumindest die beste Herleitung, die mir einfiel ...

- ▶ Klicken Sie mit rechter Maustaste auf das App-Objekt im Navigator, wählen Sie aus dem Popup-Menü den obersten Eintrag „Add to App“ und dann „Event Handler“.
- ▶ Wählen Sie dort den ersten Eintrag, „Run“, und klicken Sie auf OK.





In der Mitte des Hauptfenster ist jetzt der Code-Editor erschienen, und der Navigator zeigt als neuen Bestandteil des App-Objekts den Run-Eventhandler an. Seine Unterlegung sagt Ihnen, dass Sie ihn gerade bearbeiten.

- Geben Sie die folgenden Zeilen in den Code-Editor ein:

```
Dim MeineZahl As Integer = 10
Print MeineZahl.ToString
```

- Drücken Sie auf das Run-Symbol in der Werkzeugeiste.

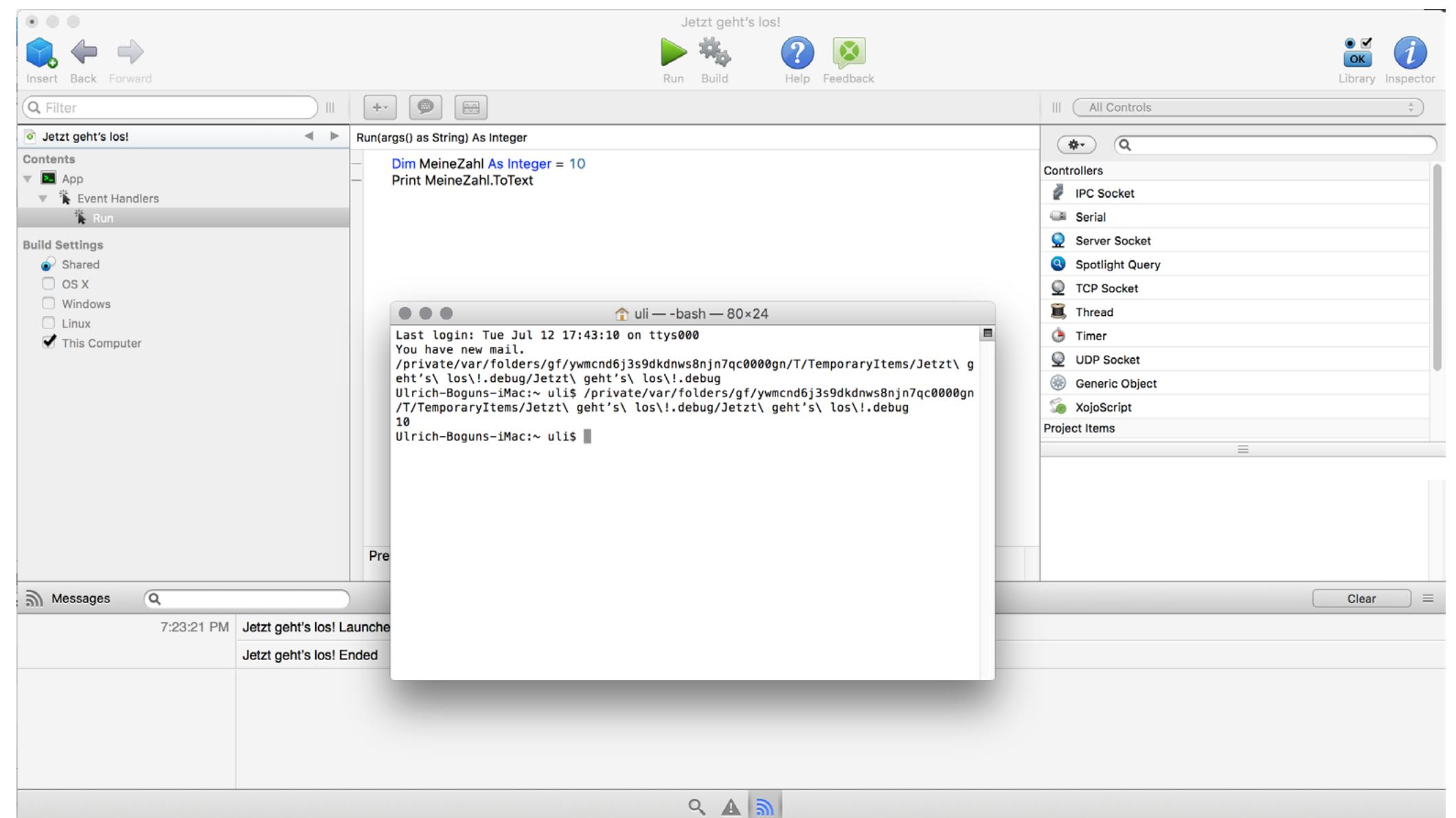
Nach sehr kurzer Bedenkzeit wird sich die Kommandozeile-anwendung Ihres Rechners öffnen und das Programm öffnen (im Screenshot sehen Sie, dass es aus irgendeinem Temporär-verzeichnis heraus gestartet wird). Zu guter letzt wirft es stolz das Ergebnis aus: 10.

Jaja, Sie haben recht: Das ist nun wirklich ein sehr unspektakuläres Programm. Sie sehen hier aber alles in Aktion, was das Programmieren ausmacht, denn

Programmieren ist das Typisieren von Speicher und das Vorgeben von Verarbeitungsschritten dieser Speicherbereiche.

Das dürfen Sie gerne das Bogunsche Theorem nennen, denn diese Definition ist auf meinem Mist gewachsen. Aber am Ende geht es wirklich immer nur darum:

- Sie reservieren Speicher und weisen ihm eine Interpretationsart zu.
- Sie machen irgendwas damit.



Der Grund, warum ich darauf so herumreite, ist jetzt weniger in meinem Bedürfnis nach 5 min. Berühmtheit zu suchen. Vielmehr haben Sie hier wieder einen potentiellen Fehler vor sich, der mit größter Wahrscheinlichkeit zu Ihren **häufigsten Fehlern** gehören wird: Die **Verwechslung von Datentypen**.

Wenn Sie mit Ihrem Nachwuchsmenschen beschließen, dass die Badewanne das Meer ist, und mitten im Spiel geben Sie da-

rin dann Autofahrgeräusche von sich, hören Sie wahrscheinlich ein „Ey Erziehungsberechtigte, bist du dumm!“ o. ä., ganz nach familiärem Umgangston.

Der Computer kann unterschiedlich reagieren. Mitunter schneidet er die Kommastellen sang- und klanglos weg, und Sie wundern sich, dass alle Rechnungen so sauber aufgehen. Oder er beendet das Spiel mit einem „Ey Programmier!“ – das nennt man dann gerne „Absturz“.



Jedenfalls dürfen Sie schon wieder Ihr Vokabelheft zücken: Es gibt was Neues! In der zweiten Programmzeile haben wir den Befehl

Print

benutzt. Falls Sie ein alter Hase sind, juchzen Sie womöglich, denn Print, das klingt nun wirklich nach Basic. Hat man früher aber damit Text auf dem Bildschirm ausgegeben, lässt sich das heutige Xojo-Print nur noch in Kommandozeilenprogrammen anwenden, denn es dient dazu, eine Zeichenkette in das Terminal auszugeben.

Eine Zeichenkette ist in Xojo vom Datentyp String (siehe Seite 27) oder moderner, wie dort schon erwähnt, vom Typ

Text

Auf die genauen Unterschiede gehen wir später noch ein. Nehmen Sie momentan einfach zur Kenntnis, dass beide zur Verwaltung von Zeichenketten taugen und dass sie untereinander kompatibel sind, zumindest können beide in den Gegenpart konvertiert werden.

Wenn Sie die letzte Seite mit gebührender Aufmerksamkeit und Hochachtung studiert haben, klingelt Ihnen meine Warnung bzgl. der verwechselten Datentypen ja sicher noch in den Ohren. **MeineZahl** ist ein Integer, aber **Print** verlangt den Datentyp Text oder String.

Weshalb jetzt auch klar sein dürfte, was

ToText

macht: Es ist eine Funktion des Datentyps Integer und konvertiert ebendiese Zahl in ihre Zeichenketten-Entsprechung – aus

MeineZahl As Integer wird TempMeineZahl As Text

Einwand gehört! **TempMeineZahl** existiert gar nicht!

Völlig richtig. Wenn wir, wie hier, einen Datentypen als Ergebnis einer Funktion zu einer anderen Funktion durchreichen (nämlich **Print**), müssen wir diese Variable nicht explizit definieren. Anders wär's, wollten wir sie vor der Ausgabe noch weitergehend manipulieren.

Ach ja: Und dann ist Ihnen sicher noch aufgefallen, dass **ToText** einfach mit einem Punkt an **MeineZahl** gehängt wurde. Das ist eine weitere Unähnlichkeit von Xojo mit althergebrachten Basic-Anleihen, nennt sich

Punktnotation bzw. Dotnotation

und hat u. a. folgenden Vorteil:

Die Umwandlung eines Integers in einen String (nicht Text also!) erfolgt über den Befehl

Str

Dabei ist **Str** keine Methode von Integer, sondern unabhängig vorhanden. Was bedeutet, Sie müssen den Befehl erst einmal kennen, um ihn nachzuschlagen. Fällt er Ihnen nicht ein, müssen

Sie relativ mühsam die Sprachreferenz durchsuchen. In Volltext sähe das dann so aus:

Print Str(MeineZahl)

► Jetzt verändern Sie aber einmal diese zweite Codezeile, so dass nur noch

Print MeineZahl.

dort steht (achten Sie auf den Punkt am Ende).

► Drücken Sie auf der Tastatur die Tabulator-Taste.

Autocomplete

liefert Ihnen eine Liste der möglichen Befehle. Das funktioniert auch mit Vorauswahl, indem Sie schon einen oder ein paar Buchstaben eingeben. Meint Xojo, einen passenden Befehl zu erahnen, blendet es an der Einfügeposition im Code-Editor eine Ellipse (...) ein als Zeichen, dass Sie per Tab eine Vorschlagsliste aussuchen können.

Konsequent integriert² bedeutet dies: Den Namen eines Datentyps eingeben. Punkt dahinter, „To“ eintippen und voilà: Die Liste der passenden Konvertierungsfunktionen!

² Wie gesagt: Xojo befindet sich noch im Umbau auf durchgängige Verfügbarkeit aller neuen Funktionen!



2.3. Ferdammte Vehler?



Viele Programmierlehrwerke stecken voller wirklich toller Anregungen zur Programmierung und lassen den Algorithmus-Adepten faszinierend fortgeschrittene Strukturen ergründen, bevor sie aber auch nur ein Wort über Fehler verlieren.

Das Ergebnis mag dann auch mal Frustration heißen, weil sich das Programm scheinbar so ganz anders verhält als im Buch versprochen – oder sagen wir mal: Es verhält sich nicht, weil der Compiler die Arbeit aus Fehlergründen verweigert.

Tatsache ist aber: Sie werden Fehlermeldungen sehr, sehr häufig zu Gesicht bekommen, und Sie sollten darob keine Vergrätzung, sondern Freude empfinden: Jede Fehlermeldung ist Ihr Freund! Statt völlig im Regen zu stehen, erhalten Sie einen Hinweis auf das Problem, der oftmals direkt zur Ursache führt (und in manchen Fällen, das sei der Fairness halber hinzugefügt, nichts weiter als große Fragezeichen im Geiste auch langjähriger Programmierprofis hervorruft).

Bevor wir eine Fehlermeldung aus dem Schlaf kitzeln: Was ist ein Bug überhaupt?

Bei [Wikipedia](#) finden Sie die Herkunftsgeschichte des Entwanzens – so jedenfalls die ungefähre Übersetzung von Debugging: Die großartige [Grace Hopper](#), der wir u.a. verdanken, dass wir heute in so klar lesbaren Sprachen wie Xojo programmieren können und uns nicht auf die Binärsprache des Computers herunterbegeben müssen, fand bei der Arbeit an einer Rechenanlage eine Motte, die zwischen den elektronischen Bauteilen des Rechners in Form eines Selbstmordattentats für einen Aus-

fall gesorgt hatte. Sie klebte die Motte in das Logbuch, zusammen mit der Bemerkung, dass hiermit das erste mal ein leibhaftiger „Bug“ – ein Käfer oder eine Wanze also eigentlich gefunden worden war. Damit hatte das Fehlerbeseitigen von Computerprogrammen seinen Namen weg: Debuggen.

 So Ihre Englischkenntnisse gut genug sind: Googlen Sie mal **Videos von Grace Hopper**. Sie sind sehr erlebenswert, und ihre Idee, Nanosekunden zu verschenken, ist eine mitreißende Physiklehrstunde, von der man sich wünschen würde, dass mehr Naturwissenschaftslehrkräfte sie kennen würden.

Jetzt aber wieder Praxis:

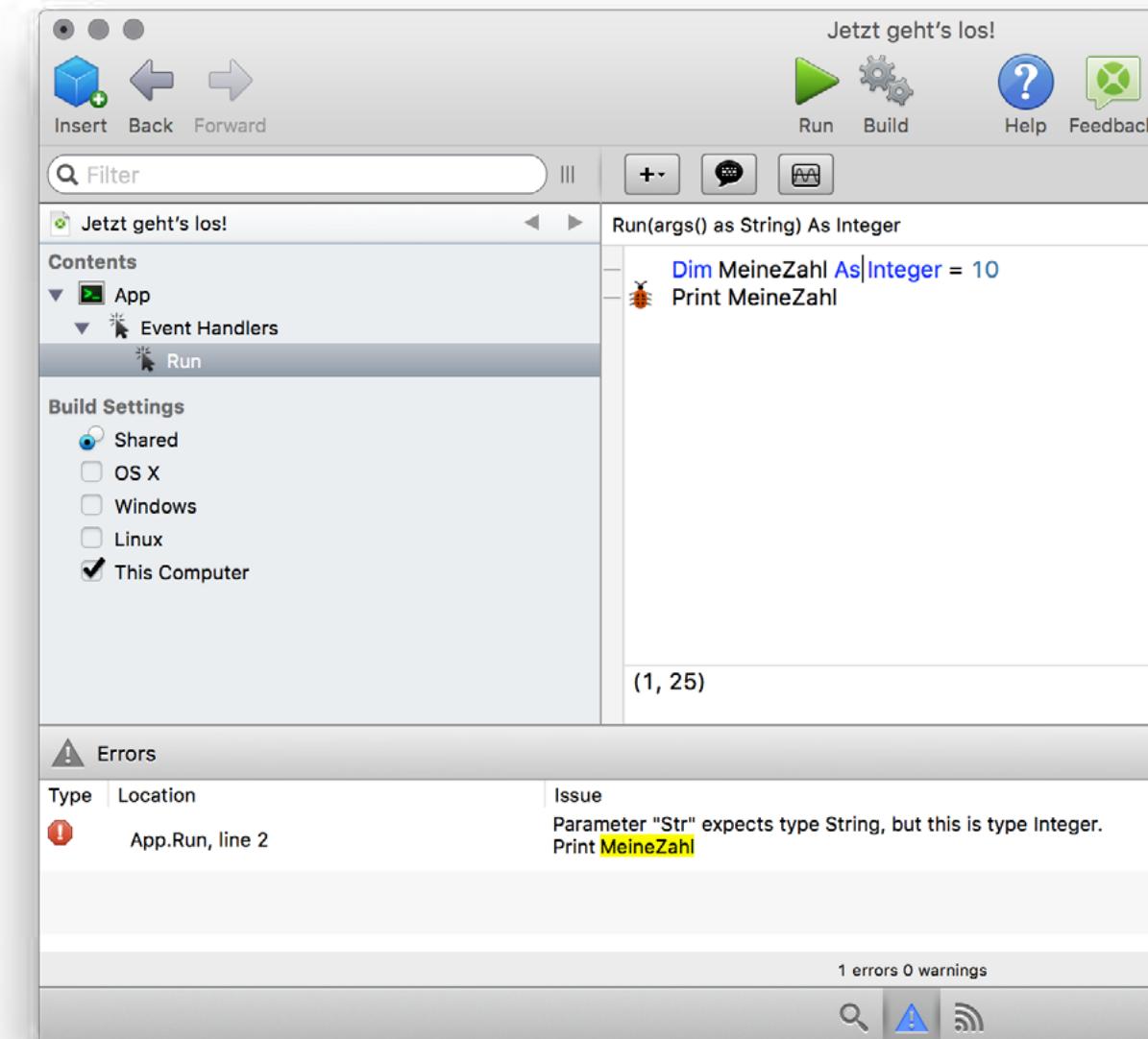
- ▶ Verändern Sie die letzte Zeile des kleinen Kommandozeilenprogramms, sodass sie

Print MeineZahl

lautet.

- ▶ Klicken Sie auf „Run“.

Das untere Panel von Xojo schaltet auf die Fehlermeldungen um und zeigt Ihnen den gefundenen Bug an, inkl. seiner Position und einer Erklärung dessen, was nicht stimmt. Die Variable **MeineZahl** wird farbig hervorgehoben, und im Code-Editor wird ein Käfersymbol an der entsprechenden Zeile angezeigt.



The screenshot shows the Xojo IDE interface. In the top right, there are buttons for 'Run', 'Build', 'Help', and 'Feedback'. The main area shows a code editor with the following code:

```
Run(args() as String) As Integer
Dim MeineZahl As|Integer = 10
Print MeineZahl
```

The 'Run' button is highlighted. Below the code editor is a 'Build Settings' sidebar with 'This Computer' selected. At the bottom, an 'Errors' panel displays a single error:

Type	Location	Issue
!	App.Run, line 2	Parameter "Str" expects type String, but this is type Integer. Print MeineZahl

There are 1 error and 0 warnings.

Wie gesagt: Nicht immer sind die Fehlermeldungen so eindeutig und leicht zu interpretieren. Häufig aber schon, und in jedem Fall erhalten Sie einen Hinweis auf die Stelle, an der es wirklich klemmt. Weitere Kammerjäger-Tricks schauen wir uns später an.



2.4. Sei nicht immer so berechnend!



Einen Speicherbereich haben wir jetzt reserviert und typisiert
– MeineZahl – aber großartig manipuliert haben wir ihn noch
nicht. Höchste Zeit, das nachzuholen!

- Lassen Sie die erste Zeile im Projekt, in der wir MeineZahl deklarieren, stehen. Löschen Sie aber den Rest im Run-Eventhandler und ersetzen Sie ihn durch

```
Dim ZweiteZahl As Integer = 99
Dim Ergebnis As Integer = MeineZahl + ZweiteZahl
Print Ergebnis.Text
```

- Klicken Sie auf „Run“, und Sie sollten jetzt das Ergebnis 109 erhalten.

Das muss ich Ihnen eigentlich nicht erklären, oder? Statt einer Variablen vom Typ Integer haben wir jetzt derer drei, und der Wert der letzten, Ergebnis, wird durch die Addition der anderen beiden Zahlen berechnet.

Ein

+

ist, ganz wie in üblicher mathematischer Schreibung, in Xojo der Befehl zur Addition.

Wo es Plus gibt, ist Minus nicht weit. Entsprechend führt eine Zeile wie

```
Dim Ergebnis As Integer = ZweiteZahl - MeineZahl
```

auch zum Resultat 89.

-

dient also der Subtraktion zweier Zahlen.

Es hat allerdings noch eine weitere Bedeutung.

- Ersetzen Sie die Ergebnisberechnung durch

```
Dim Ergebnis As Integer = -ZweiteZahl
```

Woraufhin das Resultat nun -99 lauten sollte.

- dient also auch der

Negation

einer Zahl. Es ist eine Kurzform für die Multiplikation mit -1, kehrt also das Vorzeichen um.

Sie werden wenig verblüfft sein, wenn ich Ihnen jetzt noch verrate, dass

*

zwei Zahlen miteinander multipliziert:

```
Dim Ergebnis As Integer = MeineZahl * ZweiteZahl
```

wird also 990 rauswerfen, und

/

teilt die erste durch die letzte Zahl:

```
Dim Ergebnis As Integer = ZweiteZahl / MeineZahl
```

Aber was ist hier das Resultat? 9?



Wenn Sie die Lösung dieser scheinbaren Falschrechnung nicht finden, dann schauen Sie sich noch einmal meine Ausführungen zur **Verwechslung von Datentypen** auf Seite 31 an.

Ergebnis ist ein Integer, kann also keine Kommawerte aufnehmen. Alles, was hinter dem Komma kommt, wird bei einer Umwandlung nach Integer rigoros abgeschnitten. Probieren Sie mal

```
Dim Ergebnis As Double = ZweiteZahl / MeineZahl
```



Vielleicht ein paar Kommastellen zu viel für Ihren Geschmack, aber im Großen und Ganzen viel mehr das erwartete Resultat, oder?

 Dieser Hinweis ist vor allem sinnvoll, falls Sie über Programmiererfahrung in anderen Sprachen verfügen, bei denen dies anders sein mag: Sie müssen in Xojo Zahlen, deren Berechnung ein Nicht-Integer-Ergebnis hervorbringen kann, **nicht in Fließkommazahlen konvertieren**, um eine Fließkomma-Operation an Ihnen durchzuführen.

Lustig wird es, wenn Sie die Division der Vorseite beibehalten, aber **MeineZahl** statt des Werts 10 den Wert 0 zuweisen. Lassen Sie das Programm mal laufen: Es funktioniert, aber als Ergebnis erhalten Sie

Inf

Infinity also, zu Deutsch Unendlichkeit. Leider ist der Computer keine Zen-Maschine – er führt nur die Division treuherrig aus und landet bei einer unendlich großen Zahl. (Wenn **ZweiteZahl** negativ sein sollte, übrigens auch bei negativer Unendlichkeit).

Würden Sie **Ergebnis** jetzt für weitere Rechnungen heranziehen, dann befänden Sie sich mit Ihrer Kalkulation irgendwo im metamathematischen Philosophiebereich, aber nicht mehr bei solider Zahlenjongliererei.

Wir sollten also verhindern, dass durch 0 geteilt wird. Hier, in unserer Demo, wäre das ja einfach: **MeineZahl** muss einen anderen Wert als 0 bekommen. Da wir aber auch auf Benutzeingaben reagieren können wollen, sollten wir schon ein wenig trickreicher rangehen – Stichwort „Vorbereitung für zukünftige Features“ oder „Aufwärts-Kompatibilität“ ...

Wenn also **MeineZahl** 0 ist, dann soll das Programm meckern. Ansonsten soll es die Berechnung ausführen.

Wie waren noch mal die englischen Begriffe für *Wenn* und *Sonst*?

► Ersetzen Sie den Code im Run-Eventhandler durch den folgenden (diesmal komplett, um nicht kompliziert mit Einfügezeilenangaben zu hantieren):

```
Dim MeineZahl As Integer = 0
Dim ZweiteZahl As Integer = 99
If MeineZahl = 0 Then
    Print "MeineZahl ist 0!"
Else
    Dim Ergebnis As Double = ZweiteZahl / MeineZahl
    Print Ergebnis.ToString
End If
```

► Starten Sie das Programm. Es wird sich beschweren. Ändern Sie **MeineZahl** in irgendeinen anderen Wert und starten Sie es erneut. Es wird die Division durchführen.

Der Gebrauch des Schlüsselworts

If ... Then

läutet eine

bedingte Verzweigung

ein, und diese entziffert sich schon durch einfaches Lesen:

Wenn MeineZahl den Wert 0 besitzt, dann wird der Warnhinweis ausgegeben. Ansonsten wird geteilt.

 Ein **If** kommt niemals allein! Die überprüfte Bedingung wird stets durch ein **Then** abgeschlossen und der bedingte Programmcode durch **End If** beendet – und Xojo rückt den Code dazwischen dann auch ein, um zu markieren, dass hier eine **bedingte Anweisung** steht.

Wie im Beispiel links zu sehen, können Sie aber auch weiter verzweigen. Statt einer Vielzahl von **If...End If**-Klammern ist Ähnliches möglich wie:

```
If MeineZahl > ZweiteZahl Then
    Print "MeineZahl ist größer!"
ElseIf MeineZahl < ZweiteZahl Then
    Print "ZweiteZahl ist größer!"
Else
    Print "Beide sind gleichgroß!"
End If
```

Sie können also beliebig viele Bedingungen in Reihe abfragen und, sollte keine davon erfüllt sein, zu einer umfassenden „*Ansonsten*“-Anweisung abzweigen. Dazu dienen die Begriffe

Elseif

was einmal nicht ganz klar aus dem Englischen stammt, aber ein „*sonstfalls*“ gibt das Deutsche m. E. auch nicht zum Sprachgebrauch frei. Im Gegensatz zum

Else,

das Sie je nach Vorlieben mit „*ansonsten*“, „*andernfalls*“ und jedem gleichbedeutenden Synonym übersetzen dürfen. Am Ende steht immer ein

End If.



Stillschweigend habe ich bisher Verständnis der

Vergleichsoperatoren

vorausgesetzt. Höchste Zeit, dem abzuhelfen!

Dass ein Gleichheitszeichen

=

einerseits dazu dient, einer Variablen einen Wert zuzuweisen, hatte ich bisher noch nicht explizit hervorgehoben erwähnt. Ist aber so!

Andererseits benutzt man es, um zu prüfen, ob zwei Werte identisch sind. Das sind sie aber nicht immer, und deshalb tut es manchmal not, zu schauen, ob der erste Wert kleiner als der zweite ist. Das erledigt das aus der Mathematik bekannte Symbol

<

Falls es Ihnen nicht so geläufig ist oder Sie immer Probleme hatten, sich seine Bedeutung zu merken: Stellen Sie es sich als Flüssertüte vor. Auf der schmaleren Seite spricht man leise hinein, was dann auf der breiteren Seite lauter herauskommt.

Entsprechend stellt

>

fest, ob der Wert auf der linken Seite größer ist.

Manchmal will man nicht ganz so genau sein, und es reicht, zu wissen, ob ein Wert kleiner oder gleich dem anderen ist. Das macht der Operator

<=

und, wie nicht anders zu erwarten, schaut

>=

ob der Wert auf der linken Seite größer oder gleich dem anderen ist. Sie haben recht: Eigentlich gibt es Zeichen dafür, nämlich ≤ und ≥. Die Position dieser auf der Tastatur ist aber nicht jedem geläufig.

Was auch auf ≠ zutrifft: Nicht gleich bzw. ungleich. Weshalb es auch für diese Überprüfung einen zweisymboligen Operator gibt, nämlich

<>



Vorweggesagt, falls Sie mit diesen Operatoren schon ein wenig spielen wollen: Diese Vergleichsoperatoren können Sie auf alle möglichen Datentypen anwenden, nicht nur auf Zahlen! Eine, wenn auch womöglich nicht ganz Duden-konforme, alphabetische Sortierung von Strings und Text-Variablen ist damit z. B. ebenso möglich.

An dieser Stelle ein Hinweis im Hinblick auf Ihre Gesundheit und psychische Unversehrtheit: Wenn die vorangegangenen Seiten völliges Neuland für Sie waren, setzen Sie sich damit erst einmal auseinander. Verändern Sie das kleine Beispielprogramm: Führen Sie weitere Variablen ein, lassen Sie Berechnungen durchführen, werten Sie deren Ergebnis aus und vergleichen Sie es mit Ihren Erwartungen. Kommt etwas ganz anderes raus als gedacht? Dann haben Sie vielleicht einen falschen Datentyp oder dessen Wertebereich überschritten. Machen Sie sich darum nicht zu viele Gedanken, das besprechen wir in Kürze. Auch wenn es bisher nur die Grundrechenarten sind und unser Programm über keinerlei Interaktivität verfügt: Das kommt gleich! Wenn Sie aber noch Einarbeitungszeit brauchen, dann nehmen Sie sie sich bitte. Ich warte hier solange auf Sie.

Falls Sie das alles noch einmal rekapitulieren möchten, aber nicht noch einmal lesenderweise, können Sie viele Teile des bisherigen auch in diesem Tutorial anschauen:

[Einführung in die Programmierung Teil I](#)

(das sich leider dank Adobes Festhalten an Flash hier nicht direkt einbinden lässt.)

Und, in bester Peter-Lustig-Tradition: Lassen Sie das Ganze genügend sacken. Schalten Sie auch mal ab und schauen Sie sich ein paar Bäume aus der Nähe an.

War das alles aber ziemlich kalter Kaffee für Sie, na, dann mal schnell weiter ...



2.5. **Nun hör mir doch mal zu!**



So langsam entwickeln meine Überschriften die Dynamik einer Beziehungskrise, scheint mir. Absicht ist jedoch eine ganz andre: Xojo verstehen lernen, und in diesem Abschnitt soll unser Programm uns verstehen lernen!

Bisher haben wir den Variablen `MeineZahl` und `ZweiteZahl` munter feste Werte zugewiesen. In vielen Fällen wird das auch in größeren Projekten passieren, aber noch viel häufiger soll ein Programm in der Lage sein, auf Anwenderaktionen zu reagieren.

Unser Programm sehnt sich nach Benutzereingaben, und das erledigt der Befehl, den wir in Kommandozeilenprojekten dafür benötigen:

Input

wartet auf eine Texteingabe in der Console, und ist diese geschehen und mit einem Return beendet worden, so liefert das Kommando den Text (ohne Return) als String zurück. Als Beispiel:

```
Dim s As String  
s = Input
```

oder auch wieder einzeilig

```
Dim s As String = Input
```

2.5.1. String vs. Text

ch hatte es ja schon erwähnt: String ist ein Relikt aus Basic-Tagen bzw. der Zeit, als ein jedes Zeichen eines Zeichensatzes pro Zeichen 8 Bit = 1 Byte unterzubringen war. Hier noch einmal die ASCII-Tabelle als Norm dazu.

Heute allerdings sind Zeichensätze groß. Ich meine wirklich groß. So richtig, richtig, manmachtsichjagarkeinevorstellungen-groß! Rechts sehen Sie die Zeichenübersicht-Palette einer sehr ordentlich bestückten Schrift, wie sie InDesign darstellt. Können Sie den winzigen Scrollbalken rechts oben erkennen? Dann haben Sie eine Idee, wie umfangreich ein Zeichensatz heute werden kann, und um so mehr, wenn ich Ihnen sage, dass sich hinter den kleinen Dreiecken, die Sie unter vielen Buchstaben sehen, nochmals alternative Zeichenformen verstecken!

Wie Sie's auch drehen und wenden: Mit einem Byte sind wir da maßlos überfordert! Die aus ASCII-Tagen stammende Übereinkunft 1 Byte = 1 Zeichen gilt nicht mehr. Oder nur noch sehr beschränkt: Einige Programme, Systemfunktionen, Dateiformate usw. benutzen noch pure ASCII-Codierung. Dummerweise ist in dem Uralt-Format aber nur der reine Standard-Zeichensatz Standard. Wo (falls) sich Umlaute & Sonderzeichen befinden, dafür existieren zig verschiedene Konventionen. Weiß man nicht, nach welcher Konvention sich ein dahergelaufener Text richtet, dann darf man erst einmal austüfteln, wie man statt ?? oder ♦ oder sonstigen lustigen Zeichen daraus auch wieder lesbaren Text fabriziert. Und zur Abbildung eines solch großen Zeichensatz wie rechts reichen 8 Bit eben ganz und gar nicht.



Weshalb schon vor einiger Zeit als neue Norm das

Unicode Transformation Format

eingeführt wurde, oder kurz

UTF.

Diese Norm existiert in mehreren Geschmacksrichtungen, wobei Sie vermutlich am häufigsten mit

UTF-8

zu tun haben werden.

Das ist ein sehr flexibles Format: Jedes Zeichen belegt 1 bis 4 Bytes! Und auch wenn der Xojo-String schon seit Ewigkeiten intern UTF-8 verwenden konnte, führte die alte Analogie 1 Byte = 1 Zeichen, die ja ursprünglich zur String-Konvention gehörte, immer wieder zu Problemen, da sie intern ja eigentlich nicht mehr galt – oder nur sehr eingeschränkt.

Tja, und deshalb gibt es nun den Datentyp Text, der ganz offiziell auf UTF-8 aufsetzt und Funktionen besitzt, die einem auch garantiert das richtige Zeichen ausgeben, ganz egal, wie viele Bytes es so belegt.

String wird allerdings noch auf lange Zeit vorhanden sein, und viele Funktionen benutzen dieses Datentyp auch noch – so wie etwa Input. Xojo iOS allerdings hat diese Zöpfe abgeschnitten, und die mit ihm eingeführten moderneren Datentypen halten Schritt für Schritt Einzug in die anderen Plattformen. Weshalb ich, wo irgend möglich, grundsätzlich die neuen Datentypen verwende (das Stichwort ist dann gerne „neues Framework“ oder „Xojo-Framework“), auch wenn dies auf den ersten Blick

umständlich aussehen mag. Die einfacheren Konvertierungsmethoden des neuen Frameworks machen diesen Umstand m. E. schnell wieder wett, und außerdem ist zu erwarten, dass in absehbarer Zeit Text der Standard sein wird – dann werde ich wohl nochmal viele Kapitel überarbeiten müssen.

- ▶ Ersetzen Sie den Run-Eventhandler im Beispielprojekt durch den folgenden Code:

```
Print "Erste Zahl?"  
Dim InputString As String = Input  
Dim InputText As Text = InputString.ToText  
Dim MeineZahl As Integer  
MeineZahl = Integer.Parse(InputText)  
  
Print "Zweite Zahl?"  
InputString = Input  
InputText = InputString.ToText  
Dim ZweiteZahl As Integer  
ZweiteZahl= Integer.Parse(InputText)  
  
If MeineZahl > ZweiteZahl Then  
    Print "Erste Zahl ist größer!"  
ElseIf MeineZahl < ZweiteZahl Then  
    Print "ZweiteZahl ist größer!"  
Else  
    Print "Beide sind gleichgroß!"  
End If
```

- ▶ Lassen Sie das Projekt einmal laufen und geben Sie unterschiedliche Werte ein. Schon viel mehr Programm als vorher, oder?

Sie sehen, dass ich die Variablen InputString und InputText bei der Eingabe der zweiten Zahl recycle. Ich benötige sie ja für

keine weiteren Berechnungen, also kein Grund, extra neue Variablen für ZweiteZahl einzuführen. Variablen sind ja schließlich aus gutem Grund variabel ...

Neu ist hier der Befehl

Parse,

und dieser belegt meine These für die praktische Nutzbarkeit des neuen Frameworks erneut. Parse gehört zum Datentyp Integer, erwartet einen Text und analysiert ihn auf das Auftauchen von Ziffern am Beginn(!). Sind dort welche, wird diese Zahl als Integer geliefert. Eventuelle Textanhänge, wie eine Maßeinheit, stören nicht. Gibt es keine Ziffern am Anfang, liefert Parse den Wert 0.

 Übrigens auch hier: Parse existiert nicht nur beim Integer-Typ, sondern auch bei anderen Datentypen.

Die Rechnung am Ende kennen Sie noch, es ist unverändert der Vergleich nach Größe der Werte.

 Es ist nichts falsch daran, so explizit vorzugehen wie hier. Manchmal ist dies auch das Mittel der Wahl, zumindest, um Fehlern auf die Spur zu kommen. Sie können die Eingabezeilen aber auch zusammenfassen und sich Variablenklärungen sparen – ich find's übersichtlicher:

```
Dim MeineZahl As Integer = ¬  
    Integer.Parse(Input.Totext)
```

Sollte Ihnen auffallen, dass Sie lieber auch Fließkommazahlen vergleichen wollen, ist die Umstellung dann einfach:

```
Dim MeineZahl As Double = ¬  
    Double.Parse(Input.Totext)
```



2.6. Jetzt geht's rund!



Wenn Sie das Progrämmchen jetzt ein paarmal ausprobiert haben, werden Sie mir in einer Sache bestimmt recht geben: Es ist ziemlich mühsam, es ein jedes Mal erneut zu starten, nur um zwei Zahlen zu vergleichen. Da wir jetzt Eingaben abfragen können, wäre es doch schön, wenn es so lange zur Verfügung stünde, bis ein vereinbarter Eingabebefehl es beendet.

Bis dahin soll es Warteschleifen drehen, und darum geht es in diesem Abschnitt:

Schleifen

Eine Programmschleife bedeutet die Wiederholung eines Programmcodes, bis eine bestimmte Abbruchbedingung erreicht ist. So eine Abbruchbedingung kann einfach ein Zähler sein: Die Schleife wird x-mal durchlaufen, und dann ist Schluss.

Die Bedingung kann aber auch anderer Natur sein und dabei wahlweise am Ende eines Schleifendurchgangs oder auch vorher überprüft werden: Manchmal muss man ja auch 0-x-mal irgendwo durch.

Für alle diesen Schleifentypen gibt es in Xojo Entsprechungen. Doch bevor wir uns diese anschauen, sollten wir kurz besprechen, welcher Natur so eine Überprüfung eigentlich ist.

Sie haben mit den Vergleichsoperatoren ja schon Möglichkeiten zur Überprüfung von Werten kennengelernt: =, <, >= usw. Doch in welcher Form bestimmt der Computer, ob das Prüfergebnis positiv oder negativ ausfällt, oder anders ausgedrückt,

ob die aufgestellte Behauptung `MeineZahl < ZweiteZahl` nun *wahr* oder *falsch* ist?

Er benutzt dazu einen Datentyp, der genau diese Werte enthält:

Boolean

mit den Werten

True und False



Boolean ist die feinste Entsprechung des Bits als solchem: Ist das Bit gesetzt, ist sein Wert True; ist es 0, dann ist sein Boolescher Wert hingegen False.

Auch wenn es sparsamer wäre (aber Speicher ist ja heute nicht mehr so Mangelware wie zu Heimcomputer-Zeiten): Technisch wird eine Boolesche Variable heute computerintern eher durch einen Integer abgebildet als durch ein einziges Bit, das ja einzeln gar nicht voll ansprechbar ist, sondern immer als Teil einer Verwaltungseinheit von mindestens Byte-Größe.

Viele Anekdoten um den Autodidakten George Boole, der uns die Boolesche Rechenkunst (Algebra) bescherte und damit Logik in die Mathematik brachte (Ähem. Jaja, das ist ein bisschen maßlos übertrieben!) finden Sie übrigens bei [Wikipedia](#).

Die Vergleichsoperatoren liefern als Ergebnis also immer eine Boolesche Variable. Und dank dieser kann der Computer dann eindeutige Entscheidungen treffen.

So gewappnet, können wir das Programm also umbauen:

```
Dim Antwort As Text
Do
    Print "Erste Zahl?"
    Dim MeineZahl As double = -
        Double.Parse(Input.Text)
    Print "Zweite Zahl?"
    Dim ZweiteZahl As Double = -
        Double.Parse(Input.Text)

    If MeineZahl > ZweiteZahl Then
        Print "Erste Zahl ist größer!"
    ElseIf MeineZahl < ZweiteZahl Then
        Print "ZweiteZahl ist größer!"
    Else
        Print "Beide sind gleichgroß!"
    End If

    Print "Soll ich nochmal (J/N)?"
    Antwort = Input.Text
Loop Until Antwort = "N"
```

Und dies wäre auch die erste Schleifenart in Xojo:

Do ... Loop (Until)

durchläuft eine Schleife, bis am Ende ein „N“ eingegeben wird.



Das „n“ kann übrigens auch ein kleines sein, denn standardmäßig ist der Operator = in bezug auf Zeichenketten nicht contextsensitiv, soll heißen, Groß- oder Kleinschreibung ist ihm herzlich egal.

 Was, nebenbei gesagt, für ganz Xojo gilt: Sie müssen Kommandos nicht so säuberlich mit Großbuchstaben beginnen wie ich mich hier bemühe. Und entsprechend reagiert Xojo in unserem Projekt sowohl auf „antWOrT“ wie auf „Antwort“.

Was übrigens

zulässige Variablennamen

angeht, so müssen diese mit einem Buchstaben beginnen, dürfen keine Leerzeichen, dann aber Unterstriche und Ziffern beinhalten. Und natürlich dürfen sie nicht mit Xojo-Schlüsselbegriffen identisch sein.

Leerzeilen

können Sie unbegrenzt in ihren Code einbauen. Ich mag es, zusammenhängende Abschnitte damit optisch zu markieren. Das ist aber kein Muss.

Es erschien mir für diese Art von Schleife sinnvoll, Do ... Loop zu verwenden, weil ich mit dem optionalen(!) Until die Abbruchbedingung am Ende prüfen kann.

Optional? Ja: Wenn Sie Do ... Loop ohne Abbruchbedingung, also ohne Until programmieren, haben Sie eine **Endlosschleife**, aus der das Programm niemals herauskommt, solange es läuft. (Wobei: Auch dafür gibt es Tricks. Aber das später!)

Ich könnte aber auch anders vorgehen.

► Ersetzen Sie die zweite Zeile, also Do, durch

While Antwort <> "N"

► und die letzte, Loop Until ..., durch

Wend

Wenn Sie das Programm jetzt laufen lassen, funktioniert es genau wie vorher. Warum?

Ich weise Antwort bei seiner Deklaration keinen Wert zu. Das ist in Xojo nicht nötig:

Jeder Datentyp besitzt einen definierten Vorgabewert.

Für Zeichenketten ist der Vorgabewert "", d. h., er ist leer. Und leer ist nunmal nicht gleich "N". Vor dem Ende der Schleife, vor dem Wend, wird ein neuer Wert für Antwort eingelesen. Das Programm springt dann zur While-Zeile zurück und überprüft Antwort erneut.

Und um das schriftlich festzuhalten: Eine

While ... Wend-Schleife

testet die Abbruchbedingung am Beginn, kann also genutzt werden, um eine Schleife kein- oder mehrmals zu durchlaufen. Wend ist mal wieder Computerenglisch und steht für *While-End*, also *Ende des Währends*.

Der dritte im Bunde ist die

Zählschleife

Sehr häufig weiß man nämlich, wie oft eine Schleife durchlaufen werden soll, und noch häufiger benötigt man einen Zähler, um innerhalb der Schleife etwas zu berechnen. Dafür gibt es dann

For ... Next

Hier ist die Abbruchbedingung der Zähler selbst. Klingt akademisch, daher mal ein Bericht aus der Praxis:

► Bauen Sie den Run-Eventhandler ein weiteres mal komplett um, und zwar zu

```
Dim Potenz As Integer
Print "Wievielte Potenz von 2?"
Potenz = Integer.Parse(Input.ToText)

Dim Ergebnis As UInt64 = 1
For q As Integer = 1 To Potenz
    Ergebnis = Ergebnis * 2
Next

Print Ergebnis.ToText
```

Damit haben Sie nicht nur einen Binärrechner, mit dem Sie prüfen können, ob meine Angaben zum Computer-Abakus richtig waren – Sie sehen auch eine For ... Next-Schleife am Werk.

Die Mathematik ist ganz einfach: Ergebnis wird zunächst auf 1 gesetzt, denn das ist die Potenz von 0. Dann wird ein Integer-Zähler deklariert – q –, der die Schleife Potenzmal durchläuft.



Ich könnte auch auf ein q zurückgreifen, das ich schon vorher deklariert habe. Häufig bietet sich aber diese **implizite** Art der **Deklaration** an, da man diese Zählervariablen außerhalb nicht braucht. Sie bekommen daher traditionell kleine Bezeichnungen wie q, s oder i, um einige der beliebtesten Namen zu nennen. Sie dürfen aber natürlich auch gerne jeden anderen erlaubten Namen verwenden, wenn Sie sich Interpretationen des Codes ersparen möchten. Selbsterklärende Namen sind Gold wert, vor allem, wenn Sie zu einem Projekt zurückkehren, das Sie länger nicht mehr bearbeitet haben.

Ergebnis wurde bei mir ein UInt64, also ein 64 Bit großer Integer ohne Vorzeichen, um einen möglichst großen Datenbereich abilden zu können.

Die Struktur einer For ... Next-Schleife ist auf den ersten Blick ein klein wenig komplexer als die anderen, doch das hat man schnell intus:

Sie beginnt immer mit einem

For,

gefolgt von einer Zählervariablen (die wie hier eben auch an Ort und Stelle definiert werden kann.). Dann folgt immer ein

To

oder ein

DownTo

– Sie können auch in absteigender Wertigkeit durch eine For ... Next-Schleife huschen –, und dann der Abbruchwert, nach(!)

dessen Erreichen und Schleifendurchlauf die Schleife beendet wird. Soll heißen: Eine For q As Integer = 1 To 1-Schleife läuft einmal ab!

Die Zählervariable muss vom Typ Integer (bzw. einer seiner Untertypen) oder ein Single oder Double sein.

Da dies der Fall ist, ist es naheliegend, auch mal nicht in ganzen Einstufen voran- oder bei DownTo zurückschreiten zu können. Und das geht! Ein

```
For q As Double = 0 to 1 Step 0.1 ... Next
```

geht in Schritten von 0.1 durch die Werte von 0 bis 1, und ein

```
For q As Integer = 100 DownTo Endwert Step 2
...
Next
```

bewegt sich in 2er-Sprüngen abwärts von 100 bis zu einem variablen Endwert.

Noch gar nicht erzählt und deshalb mal erwähnenswert: Da Xojo eine Englisch-basierte Programmiersprache ist, müssen Sie **Fließkommawerte** in US-Manier **mit** einem **Punkt statt** eines **Kommas** einprogrammieren. Die Ein- und Ausgabe kann allerdings automatisch immer den Spracheinstellungen des Rechners gemäß erfolgen.

Freiflug! Das war jetzt wieder eine ganze Menge, deshalb fühle ich mich zu folgender Nerverei genötigt: Spielen Sie wieder mit dem erworbenen Wissen! Basteln Sie kleine Rechnereien mit und ohne Schleifen, experimentieren Sie auch mit Zeichenketten. Dazu noch eine Ergänzung:

+

funktioniert auch mit String und Text! Probieren Sie mal etwas wie

```
Dim Erkenntnis As Text = "Xojo ist toll! "
For q As Integer = 1 to 10
    Erkenntnis = Erkenntnis + Erkenntnis
Next
Print Erkenntnis
```

Übertreiben Sie es aber nicht mit dem Endwert – das Terminal ist für allzu riesige Texte en bloc nicht gedacht!

Klopfen Sie einfach auf das Display, wenn Sie ausgespielt haben. Dann treffen wir uns auf der nächsten Seite.



2.7. Programmieren mit Methode(n)

Haben Sie ein paar erfolgreiche Variationen hinbekommen? Sehr schön! Dann lassen Sie uns jetzt mal etwas Struktur in die Sache bringen. Kehren wir zurück zu unserem Zahlenvergleich und erweitern ihn ein wenig:

- ▶ Ersetzen Sie den Code im Run-Eventhandler durch folgenden:

```
Dim Antwort As Text
While Antwort <> "N"
    Print "Erste Zahl?"
    Dim MeineZahl As Integer = ¬
        Integer.Parse(Input.Totext)

    Print "Zweite Zahl?"
    Dim ZweiteZahl As Integer = ¬
        Integer.Parse(Input.ToText)

    If MeineZahl Mod 2 = 0 Then
        Print "Erste Zahl ist gerade"
    End If

    If ZweiteZahl Mod 2 = 0 Then
        Print "Zweite Zahl ist gerade"
    End If

    Print "Nochmal? (J/N)"
Wend
```

Neu ist hier der Operator

Mod

Mod ist kurz für Modulo, eine mathematische Funktion, die im Schulunterricht meistens nicht explizit erwähnt wird, aber in schriftlichen Divisionen immer wieder Erwähnung findet:

$11 / 2 = 5 \text{ Rest } 1$

Und dieser Rest, als Ganzzahl, das ist das Ergebnis einer Modulo-Operation: Sie berechnet den Rest, der bei der Division zweier Integers übrigbleibt.

Entsprechend kann EineZahl Mod 2 feststellen, ob EineZahl gerade oder ungerade ist. Bleibt kein Rest, konnte glatt durch 2 geteilt werden, und das ist das Kennzeichen einer geraden Zahl.

Wenn Sie sich den Programmcode einmal genauer anschauen: So kurz er ist, er strotzt vor Wiederholungen, oder? Und wenn Sie sich vorstellen, dass wir vielleicht noch weitere Zahl-Variablen einführen oder weitere, sich womöglich ebenso wiederholende Auswertungen integrieren, dann wird es bald Zeit, die Tomatensauce aufzukochen, denn wir bewegen uns auf einen Spaghetticode zu: Mit zunehmender Codelänge sinkt die Lesbarkeit, und außerdem tippen wir viel zu viele Code-Duplikate. Und, noch gravierender: Stellen Sie sich vor, wir stellen später fest, dass unser Algorithmus zum Überprüfen der Gradzahligkeit einen Fehler hat. Dann müssen wir jede seiner Fundstellen im Code durch die neue Version ersetzen.

Deshalb sollten Sie gar nicht erst so anfangen. Auch wenn wir noch gar nicht beim objektorientierten Programmieren angekommen sind: **Strukturiert** programmieren sollten wir von Anfang an. Falls Sie erpicht auf eine neue Vokabel sind (aber keine Angst; Sie werden nicht abgefragt):

Prozedural

wäre dies. Und zwar in der Definition der Zerlegung eines Problems in seine Teilespekte. (Prozedural wird auch als Gegenstück zum objektorientierten Programmieren genannt und nein, da wollen wir schließlich hin!)

Sie können in Xojo jederzeit und beliebig Programmcode „auslagern“, was sich natürlich besonders bei Programmabschnitten anbietet, die Sie mehrfach im Code verwenden. Dann steht dieser Programmabschnitt auch nur einmal in Ihrem Programm, und sollten Sie später etwas ändern müssen, tun Sie dies auch nur an dieser einen Stelle. In Xojo wird so eine Codeauslagerung u. a. durch

Methoden

ermöglicht.

Eine Methode ist, kurz und vereinfacht ausgedrückt, ein wieder verwendbarer Programmschnitt, der unter seinem Namen aufgerufen werden kann. Eine Methode kann dabei Variablen entgegennehmen und optional auch eine Variable zurückliefern.



 Wieder ein Hinweis für Programmiersprachumsteiger:
Viele andere Sprachen unterscheiden hier zwischen Methoden und **Funktionen** und definieren Methoden als die Programmabschnitte, die keine Resultate zurückliefern, während die Funktion sich durch eine Rückgabe-Variable auszeichnet. Xojo unterscheidet intern zwar ebenso, vereinfacht dem Programmierer dies aber, indem von ihm alles als Methode anzulegen ist. Wird ein Rückgabewert definiert, so macht Xojo intern eine Funktion daraus.

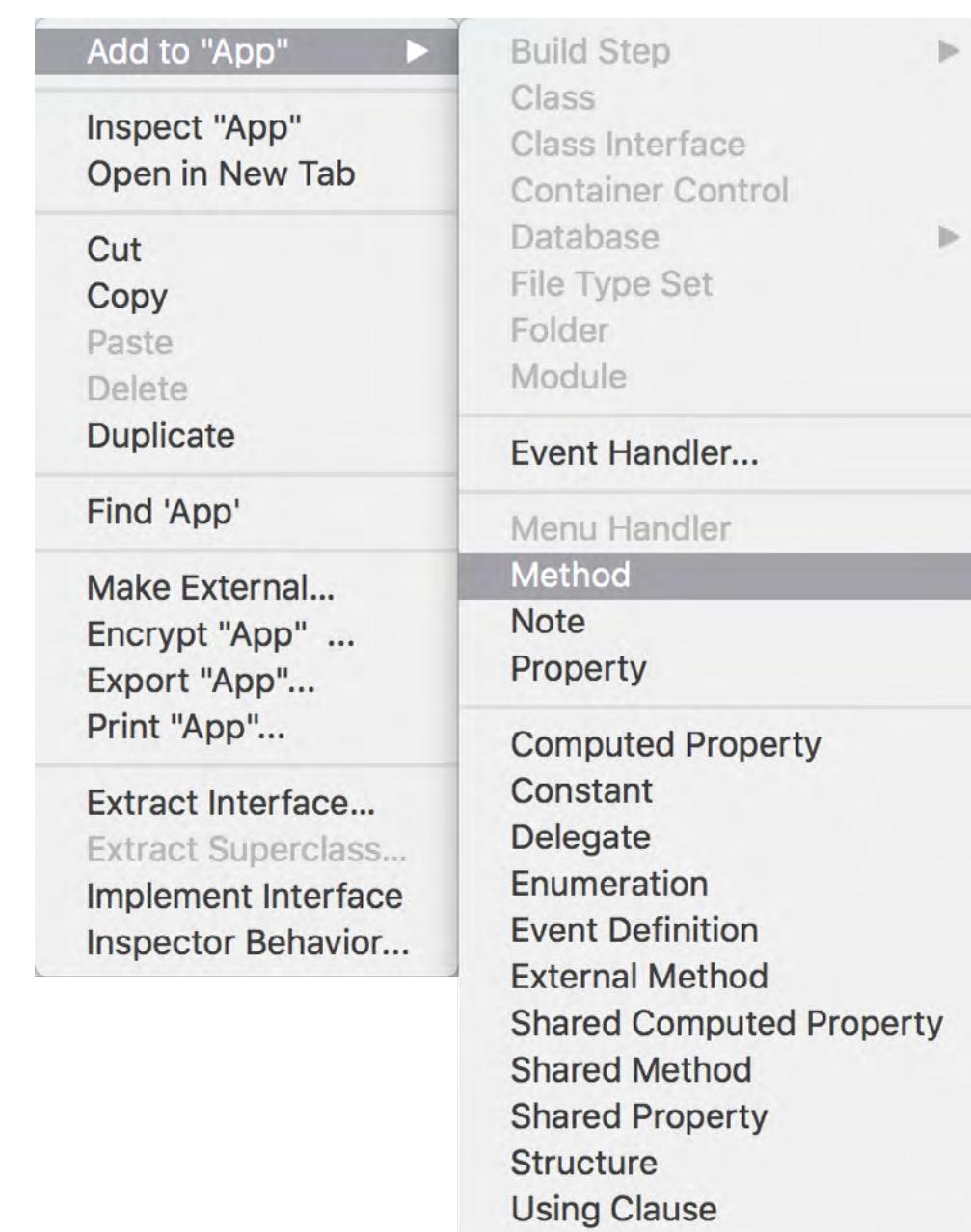
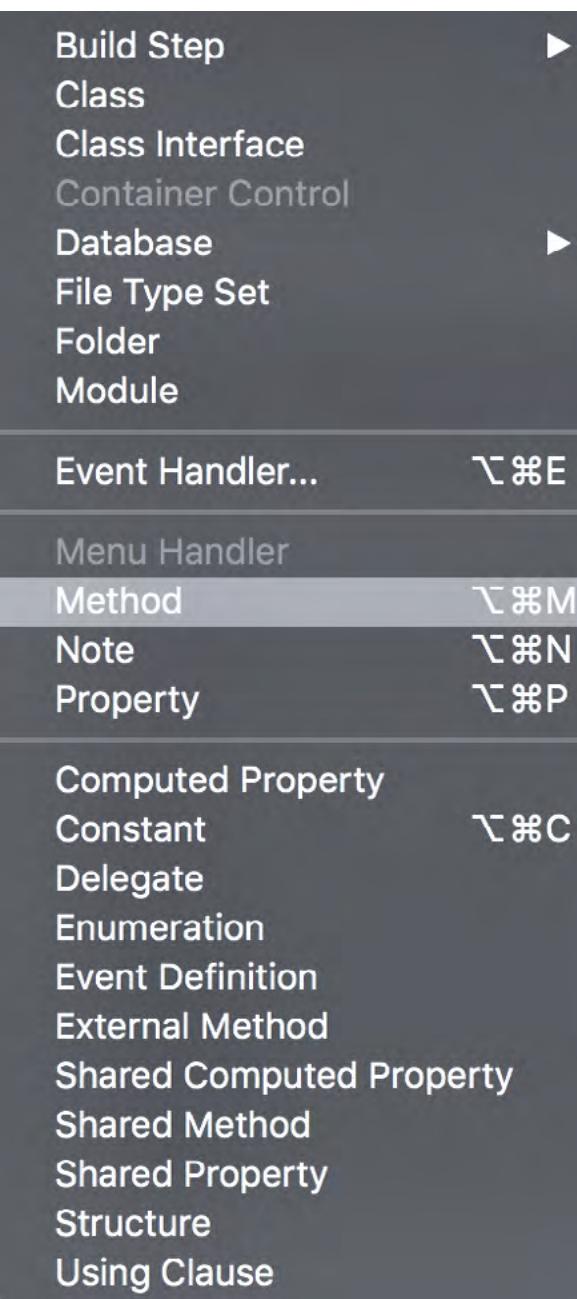
► Klicken Sie im Navigator auf das App-Objekt, sodass es ausgewählt ist.

 Gleich noch ein Hinweis dazu: Xojo ist recht komfortabel, was den bevorzugten Arbeitsstil angeht. Für viele Funktionen gibt es mehr als eine Möglichkeit der Hervorrufung. Ich möchte Ihnen nicht meinen Arbeitsstil aufzwingen und deshalb ab jetzt beginnen, neue Features in dieser Hinsicht möglichst umfassend zu erklären. Danach finden Sie nur noch Anweisungen der Form „Fügen Sie ... ein“, und Sie machen das dann so, wie es Ihnen am meisten behagt, ok?

2.7.1. Methoden einfügen

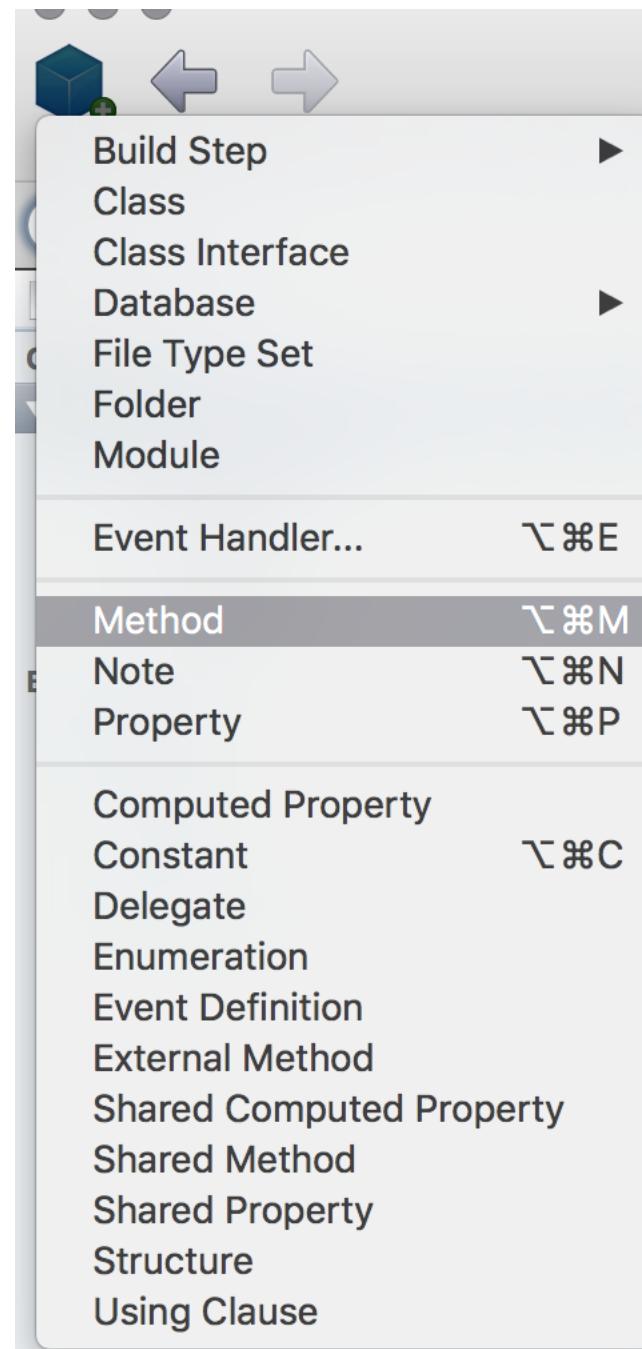
Eine Methode fügen Sie in das ausgewählte Objekt, indem Sie

- in der Menüzeile unter dem Eintrag „Insert“ den Punkt „Method“ auswählen
- das entsprechende Tastaturkürzel benutzen
- nach Rechtsklick auf das Objekt den Punkt „Add to (Objekt)“ anwählen und aus dem Sekundärmenü dann „Method“

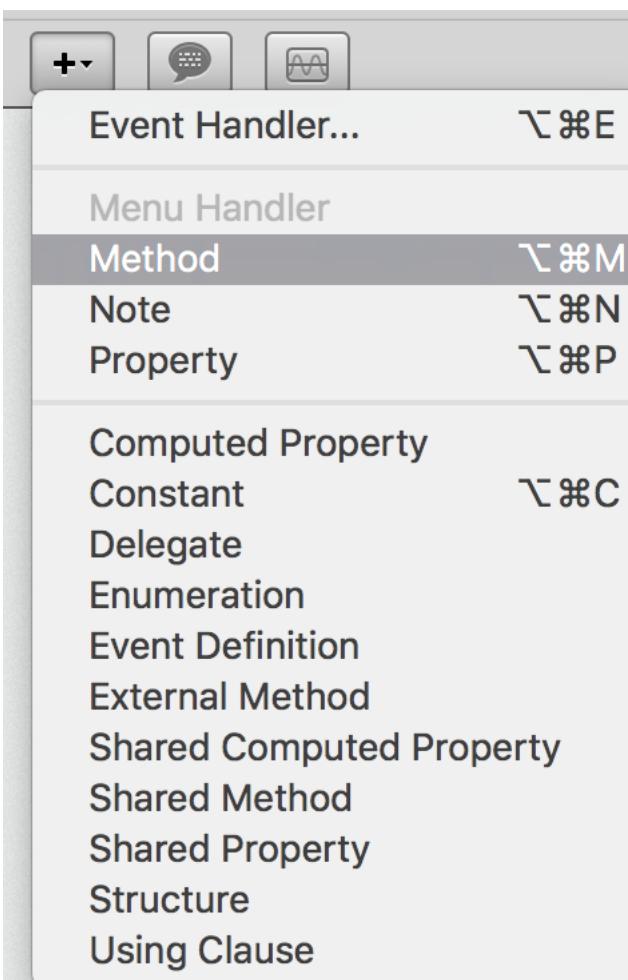




- Auf das „Insert“-Symbol in der Werkzeugeiste klicken und dann den Eintrag „Method“ anwählen

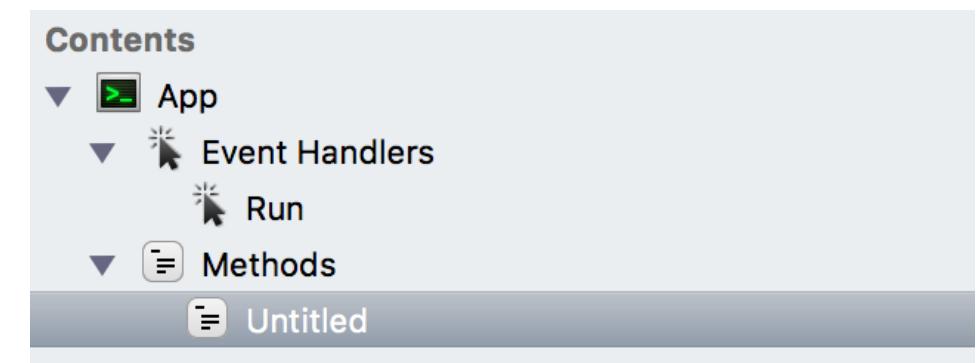


- oder auf das Einfügen-Symbol im Code-Editor klicken und den Eintrag „Menü“ selektieren



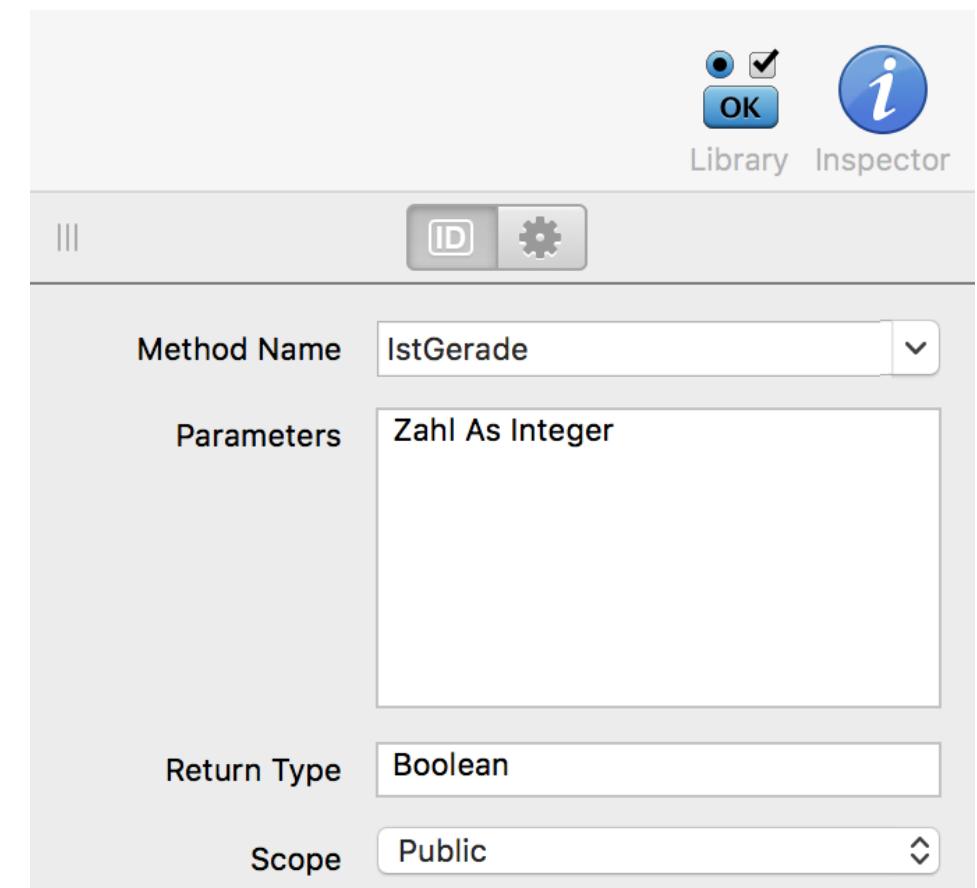
- Fügen Sie nun also eine neue Methode ins App-Objekt ein.

Im Navigator erscheint nun ein neues Objekt im App-Objekt, nämlich die unbenannte Methode:



und im Inspector wird sie ebenso angezeigt.

- Geben Sie die untenstehenden Werte in den Inspector ein:





► Der Code der Methode selbst besteht aus:

```
If Zahl Mod 2 = 0 then
    Return True
else
    Return False
end If
```

Der neue Begriff

Return

ist wieder einmal sehr selbsterklärend: „Gib zurück“. Der zurückgelieferte Datentyp muss dem entsprechen, den Sie als Rückgabewert im Inspector definiert haben – hier also Boolean.

Man kann einer Methode also Parameter übergeben (muss aber nicht), die dann unter dem in der Methodendeklaration stehenden Namen bearbeitet oder ausgewertet werden können. In unserer IstGerade-Methode wäre das der Parameter Zahl.

Somit haben wir die Mathematik-Fähigkeiten unseres Projekts erweitert. Wollen wir feststellen, ob ein Integer gerade oder ungerade ist, reicht dafür ein Aufruf der Art

```
Dim Gerade As Boolean = IstGerade (eineZahl)
```

Der Code ist hier zur besseren Lesbarkeit sehr explizit ausformuliert. Eine Methode mit Rückgabewert (also streng genommen **eine Funktion**) **liefert immer einen Wert zurück**. Wird dieser nicht durch ein Return definiert, dann greift Xojo zum Standardwert des Datentyps. Bei Boolean ist das False. Es reicht also auch völlig,

```
If Zahl Mod 2 = 0 Then
    Return True
End If
```

zu schreiben.

Und es geht sogar noch kürzer. Einerseits können Sie einzeilige If...End If-Bedingungen auch einzeilig schreiben:

```
If Zahl Mod 2 = 0 Then Return True
```

In diesem Fall entfällt das End If, es wird sozusagen vom Return ersetzt, das Sie am Ende der Zeile eingeben.

Falls Sie es lieber ausführlich mögen, geht das mit einzelnen Entscheidungen ebenso:

```
If Zahl Mod 2 = 0 Then Return True Else -
    Return False
```

(Nur bekomme ich das hier im Satz nicht in eine Zeile.) Auch hier entfällt das End If.

Und nun noch die meiner unbescheidenen Meinung nach eleganteste Methode: Wenn Sie sich den Ausdruck Zahl Mod 2 = 0 anschauen: Dieser resultiert ja schon in einem Booleschen Wert, soll heißen, Zahl Mod 2 = 0 ergibt True, wenn wir eine gerade Zahl haben, und False, wenn nicht. Da ist es ja eigentlich doppelt gemoppelt, das Ergebnis abzufragen und dann ebenjenen Wert erneut zu erzeugen. Geben wir doch einfach den Booleschen Wert der Auswertung selbst zurück:

```
Return Zahl Mod 2 = 0
```

Wie gesagt: Alle diese Varianten resultieren in der exakt gleichen Funktionalität. Es mag sein, dass die ausführlichen Arten in ein klein bisschen ineffizienterem Code resultieren. Wenn Ihnen die Lesbarkeit einer expliziteren Ausführung aber lieber ist: Nur zu!

Fertig? Fast! Die Funktion wird so funktionieren, das kann ich Ihnen garantieren (und das überprüfen wir auch gleich). Stellen Sie sich aber mal vor, Sie haben ein wirklich großes Projekt erarbeitet und eine Vielzahl eigener Methoden untergebracht. Natürlich haben Sie den Funktionen sehr beredte Namen verliehen. Trotz allem mag es manchmal schwierig sein, sich die genauen Features zu merken. Dann immer wieder in den Code schauen, um sicherzugehen, ob man die richtige Methode ausgewählt hat? Die ganze Zeit schon eine Dokumentation schreiben und konsultieren?

Letzteres ist keine schlechte Idee, nur das kontinuierliche Nachschlagen mag irgendwann nerven. Lassen Sie die Funktion für sich selber sprechen!



2.8. Keine Ruhe auf den billigen Plätzen!

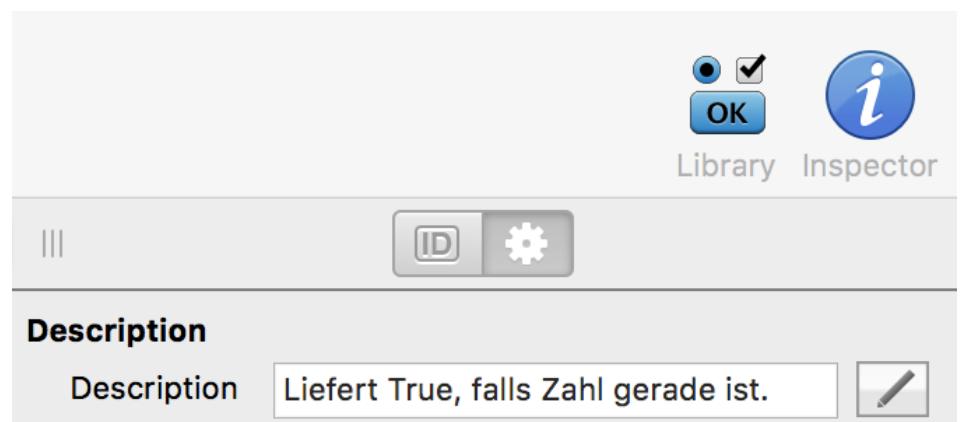
- Achten Sie darauf, dass die `IstGerade`-Methode im Navigator weiterhin angewählt ist, und klicken Sie im Inspector (ggf. also vorher auf das Inspector-Symbol, um ihn statt der Library anzuzeigen) auf das kleine Zahnradsymbol rechts oben.

Der Inspector blendet auf ein zweites Panel um. Hier interessiert uns gerade nur ein Feld, nämlich die

Description,

also die Beschreibung. Diese ist der Text, der im unteren Feld des Code-Editors auftaucht, wenn Sie den Mauszeiger über einen Befehl Ihres Codes platzieren.

- Geben Sie in das Description-Feld einen wohlfeilen Text ein, der in knappen Worten beschreibt, was Ihre Funktion so tut. Etwa so wie hier:



Wenn Sie möchten, können Sie auch auf das Stift-Symbol rechts neben dem Feld klicken und den Text beliebig erweitern – es geht dann ein kleines Texteditor-Fenster auf, in dem Sie auch

Return verwenden können, um Absätze zu erzeugen. Allerdings wird allzu langer Text dann zu einer Scrollorgie, weil die Description in nur wenigen Zeilen angezeigt wird. Ein Feature-Vorschlag von mir, das Feld größenveränderbar zu machen, hat noch keine Aufnahme in den Xojo-Kanon gefunden. Eventuelle Beschreibungen der zu übergebenden Parameter aber sind im Fall des Falles gut investierte Autorentätigkeiten.

Wo wir gerade bei Hilfestellungen für das spätere Selbst sind: Tun Sie sich den Gefallen und machen Sie regen Gebrauch von Kommentaren. Wo immer Sie eine komplexe Lösung ausgetüftelt haben, wo immer Sie sich selbst Notizen für spätere Erweiterungen hinterlassen wollen, eigentlich überall, wo sich der Sinn nicht sofort durch Lesen des Quelltexts ergibt: Hinterlassen Sie einen

Kommentar!

Das können Sie in Xojo, indem Sie im Code-Editor entweder hinter ein Stück Quellcode oder in eine neue Zeile tippen, und zwar beginnend mit

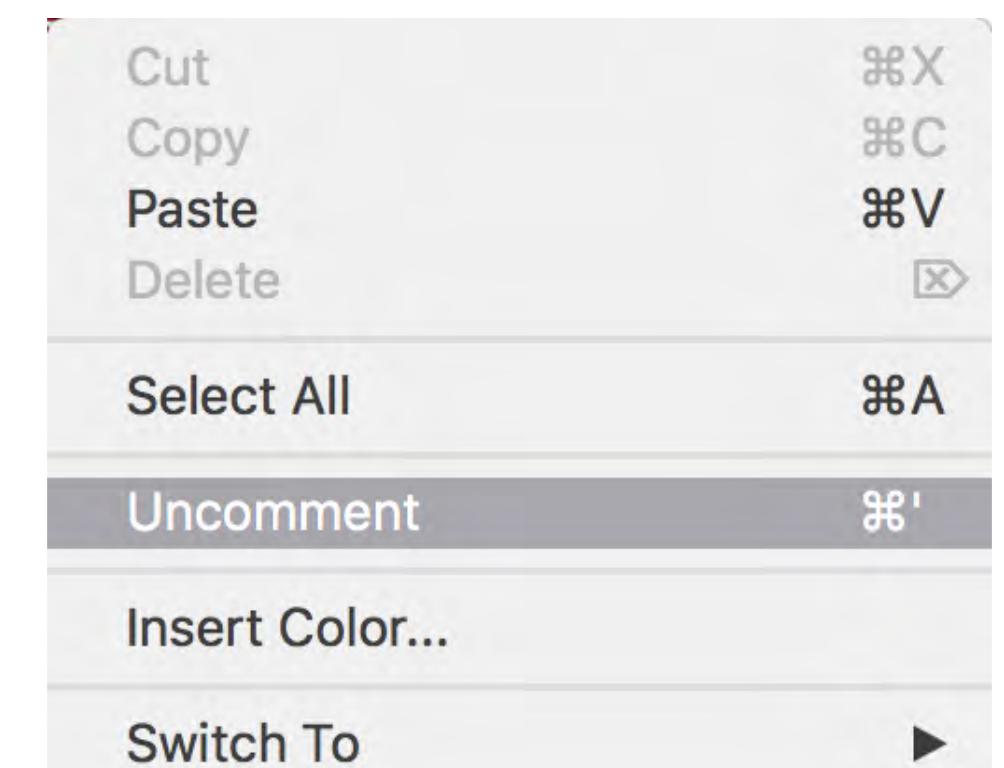
', // oder REM

Wobei sich REM wieder vortrefflich aus dem englischen Remark, also Anmerkung, übersetzen lässt.

Alles, was hinter einem der obigen Symbole steht, wird von Xojo als Kommentar angesehen, entsprechend farbig hervorgehoben und beim Kompilieren nicht berücksichtigt – Sie müssen sich

also keine Sorgen machen, dass das ausführbare Programm von reichlichen Kommentaren aufgeblättert werden könnte. Wenn das kein guter Grund ist, reichlich zu kommentieren ...! Sie werden es sich später jedenfalls danken.

Wenn Sie im Code-Editor den Cursor in einer Zeile positioniert haben, können Sie übrigens auch per Rechtsklick ein Pop-Up-Menü aufrufen, mit dem Sie u. a. eine Zeile bequem kommentieren oder de-kommentieren können. **Somit lässt sich während der Testphase auch mal Code ausschalten, ohne ihn ganz löschen zu müssen.** Den gleichen Comment/Uncomment-Eintrag finden Sie auch im Edit-Menü.





Und nun nochmal zurück zu unserem Programm: Wir haben jetzt eine Funktion, die uns eine Boolesche Variable liefert mit der Aussage, ob unsere Zahl eine gerade ist. Die Redundanz im Programm haben wir aber noch nicht entfernt: Der „Auswertungstext“ wird nach wie vor doppelt ausgegeben, für jede Zahl separat – mal abgesehen davon, dass wir unsere Funktion noch nicht integriert haben.

Da es mir ja gerade um das Vermeiden doppelten Programm-codes ging, hilft an dieser Stelle nur: eine weitere Methode!

Wenn es von jetzt an um das **Einfügen von Methoden** geht, werde ich auf erklärende Bildchen verzichten. Stattdessen bilde ich die Methode komplett ab, inkl. ihrer Übergabe- und ggf. Rückgabeparameter. Das sieht dann so aus:

```
Sub GeradeZahlAuswerten(Zahl As Integer, ¬  
    Zahlname As Text)  
    If IstGerade (Zahl) Then  
        Print Zahlname + " ist gerade"  
    else  
        Print Zahlname + " ist ungerade"  
    End If  
End Sub
```

Das Praktische daran: Sie können diesen Code in die Zwischenablage kopieren und ihn, wenn Sie darauf achten, dass das App-Objekt im Navigator aktiv angewählt ist, einfach einfügen – bzw. fast einfach, denn Sie müssen die erste Zeile am **¬** zu einer zusammenfügen. Machen Sie es sich nach Möglichkeit nicht zu einfach: Kopieren Sie gerne die Texte, aber stellen Sie immer sicher, dass Sie jede Zeile verstehen. Falls nicht, schlagen Sie im Index die unbekannte Funktion nach – oder was immer Ihnen gerade schwammig erscheint.

Wenn Sie zwecks verbessertem Lernerlebnis lieber selbst tippen, folgen Sie der Anleitung zum Methodeneinfügen auf Seite 43.

Sub und End Sub

lassen Sie dabei außer Acht: Das ist die Kennung für eine Methode, die Xojo für Sie anlegt, sobald Sie eine Methode einfügen. Definieren Sie einen Rückgabewert, stehen an diesen Stellen dann

Function und End Function.

Geben Sie nur den ersten Begriff nach Sub (bzw. eben Function) als Methodennamen ein – hier also GeradeZahlAuswerten. Was dann in Klammern folgt, sind die Übergabeparameter – diesen Text (also Zahl As Integer, Zahlname As Text) müssen Sie ergo in das Feld „Parameters“ des Inspectors eintragen.

Und wenn ich Ihnen eine Funktion auf diese Art zum Kopieren kredenze, finden Sie hinter dieser Klammer noch den Rückgabeparameter. Der gehört entsprechend in das Feld „Return Type“.

Die abschließende Zeile, End Sub bzw. End Function, können Sie wieder ignorieren. Die erstellt Xojo unsichtbar selbstständig.

Damit Sie sich davon ein Bild machen können: Das ist die Funktion IstGerade als kopierbarer Programmcode – bzw. eben als eine mögliche Variante:

```
Function IstGerade(Zahl As Integer) As Boolean  
    Return Zahl mod 2 = 0  
End Function
```

- Fügen Sie also die Methode GeradeZahlAuswerten in das App-Objekt ein.

Ich denke, diese neue Methode bedarf keiner weiteren Erklärung – es ist ja im Großen & Ganzen das, was wir bisher im Run-Eventhandler gemacht haben. Nur übergebe ich sowohl die Zahl als auch ihren Namen, der dann in der Auswertung mit dem entsprechenden Text verknüpft wird.

- Verändern Sie den Code des Run-Events, sodass er wie folgt lautet:

```
Dim Antwort As Text  
While Antwort <> "N"  
    Print "Erste Zahl?"  
    Dim MeineZahl As Integer = ¬  
        Integer.Parse(Input.Totext)  
  
    Print "Zweite Zahl?"  
    Dim ZweiteZahl As Integer = ¬  
        Integer.Parse(Input.ToText)  
  
    GeradeZahlAuswerten (MeineZahl, ¬  
        "Erste Zahl")  
    GeradeZahlAuswerten (ZweiteZahl, ¬  
        "Zweite Zahl")  
  
    Print "Nochmal? (J/N)"  
Wend
```

- Lassen Sie das Programm ausführen, erfreuen Sie sich daran, und dann erfinden Sie bitte selbst ein paar Methoden (z.B. die Größer-/Kleiner-Auswertung, aber seien Sie viel kreativer!). Wenn alles klappt: Gönnen Sie sich eine Pause; die Einführung ist gleich geschafft!



2.9. Was deins ist, ist auch meins, aber was meins ist, ...

... geht dich gar nichts an.“ Das war eine der Weisheiten, mit denen meine Schwester mich ins Familienleben integrieren wollte, nebst „brüderlich teilen, schwesterlich bescheißen.“

Keine Angst! Ich will mich nicht bei Ihnen auf die Couch legen. Vielmehr ist ihre erste Aussage eine feine Umschreibung des finalen Themas: Es geht um den

Scope.

Das ist, glaube ich, eine uns aus dem Schulunterricht weniger vertraute Vokabel. Als Übersetzungsvorschlag wird eine ganze Reihe von Begriffen [geliefert](#), von Spielraum über Betätigungs-feld bis zu Einflussbereich und Entfaltungsmöglichkeiten. Wikipedia spricht im Programmier-Zusammenhang dann auch vom [Sichtbarkeitsbereich von Variablen](#). Und das trifft alles zu. Je nach Betrachtungsweise könnten wir auch vom Lebensraum oder dem Lebenszyklus einer Variablen sprechen. Denn darum geht es hier.

Wenn Sie sich nochmal meine Definition des Programmierens auf Seite 31 anschauen: Der erste Schritt besteht ja immer im Reservieren und Typisieren von Speicher. Stimmt, wir haben heute viel mehr Arbeitsspeicher zur Verfügung, als wir uns das zu Atari-Zeiten hätten träumen lassen. Allerdings fressen die Betriebssysteme alleine schon ein riesiges Mehrfaches der früheren Computer-Kapazitäten, wir können zig Programme gleichzeitig laufen lassen, und diese sind mitunter auch schon riesengroß oder können ihrerseits enormen Speicherhunger entwickeln. Soll heißen:

Arbeitsspeicher ist ein rares Gut. Egal wieviel Sie davon verbaut haben: Wird der Rechner spürbar langsamer, weil er Speicher auf die Festplatte auslagert, dann weiß man, es hätte mehr sein dürfen.

Nehmen wir mal an, unser bisheriges Programm würde viel, viel häufiger die `GeradeZahlAuswerten`-Methode aufrufen. Bei jedem Aufruf werden zwei Variablen deklariert, nämlich `Zahl` und `Zahlname`. Denn das ist, was der **Parameters**-Block einer **Methode** macht: Es werden neue Variablen angelegt und mit dem Übergabewert gefüllt. Sie könnten in der `GeradeZahlAuswerten`-Methode den Parameter `Zahl` nach Herzenslust verändern, ohne dass dies eine Wirkung auf `MeineZahl` oder `ZweiteZahl` hätte, von denen sein ursprünglicher Wert stammt.

Würde diese Methode jetzt also sehr häufig aufgerufen und jedesmal neuer Speicherplatz für die Variablen reserviert werden: Keine Frage, dass der RAM-Bedarf Ihres Programms irgendwann astronomische Höhen erreichen würde.

Weshalb das so auch nicht gemacht wird. Vielmehr werden Variablen im Arbeitsspeicher

dynamisch

verwaltet. Das heißt, ebenso wie sie on demand reserviert werden, also dann, wenn sie benötigt werden, werden sie auch wieder entsorgt, wenn man sie nicht mehr braucht. Somit funktioniert es dann auch, dass immer wieder Variablen der Namen

`Zahl` und `Zahlname` in der Methode `GeradeZahlAuswerten` deklariert werden können, ohne dass sich irgendetwas beift. Wird diese Methode aufgerufen, sind eventuell früher angelegte Variablen dieser Namen bereits dem digitalen Erdboden übergeben worden: Sie existieren für den Computer nicht mehr und sind wieder dem frei verfügbaren Speicher hinzugefügt worden. Ob in der Praxis neu deklarierte Variablen wieder genau denselben Speicherbereich okkupieren wie ihre Vorgänger, kann uns als Programmierer völlig egal sein (und wird in der Praxis auch selten der Fall sein, nicht zuletzt, weil unser Programm nicht das einzige ist, das laufend Speicher verlangt und wieder freigibt).

Um mal wieder mit Fachbegriffen um mich zu werfen: Die Variablen `Zahl` und `Zahlname` sind

lokale Variablen

der Methode `GeradeZahlAuswerten`.

Wieder einmal: So trivial dieser Umstand klingen mag: In der Praxis gehört sein Übersehen zur Top 10 der beliebtesten Programmierfehler. Denken Sie immer daran:

Jeder Programmbestandteil hat eine definierte Lebenszeit – seinen Scope.

Und das trifft auf alles zu, was Sie im Navigator eines Xojo-Projekts sehen können. Anfänglich wurde uns dort ja ein App-Objekt hineingestellt. Nun, das lebt natürlich so lange im Arbeitsspeicher, wie das Programm selbst ausgeführt wird.



Innerhalb der Programmstrukturen gilt dieser Umstand genau so.

Der Einrückungsgrad von Programmstrukturen gibt Aufschluss über den Scope seiner Elemente.

Das klingt wieder mal sehr akademisch, daher eine Klarstellung: Schauen Sie sich noch einmal die Methoden- und Funktionsabbildung auf Seite 47 an. Durch die – in Xojo eigentlich nicht sichtbaren, bzw. grafisch durch das Methoden-Objekt im Navigator dargestellte Sub/End Sub- bzw. Function/End Function-Klammer ergibt sich eine Einrückungsebene für den ganzen zur Methode gehörigen Quelltext.

Alles, was nun in dieser Ebene steht, ist auch nur vorhanden, während die Methode selbst durchlaufen wird. Außerhalb der Methode GeradeZahlAuswerten existieren die Variablen Zahl und ZahlnName nicht!

Man kann sich wie gesagt ganz prima selbst Programmierbeinchen stellen, wenn man diesen Umstand übersieht. Ein Beispiel:

- ▶ Ersetzen Sie den Code im Run-Eventhandler durch diesen hier:

```
Do
    Print "Erste Zahl?"
    Dim MeineZahl As Integer = ¬
        Integer.Parse(Input.Totext)

    Print "Zweite Zahl?"
    Dim ZweiteZahl As Integer = ¬
        Integer.Parse(Input.ToText)

    GeradeZahlAuswerten (MeineZahl, ¬
        "Erste Zahl")
    GeradeZahlAuswerten (ZweiteZahl, ¬
        "Zweite Zahl")

    Print "Nochmal? (J/N)"
    Dim Antwort As Text
    Antwort = Input.ToText
    loop until Antwort = "N"
```

Wenn Sie versuchen, das Programm auszuführen, werden Sie von einem Käfer heimgesucht. Xojo vermerkt einen Fehler an der letzten Programmzeile, und zwar dergestalt, dass „Antwort“ ihm völlig unbekannt wäre:

This item does not exist.

Sie sehen kleine farbige Linien im Quelltext, die den Scope des Run-Eventhandlers selbst angeben: In blau die Do ... Loop Until-Schleife und die kurze lila Linie als Lebenszeit von Antwort. Selbst wenn Sie Dim Antwort As Text unter die Do-Zeile verschoben: Damit würde zwar seine Lebenszeit ver-

längert, aber nicht so weit, dass Antwort der umgebenden Do ... Loop Until-Schleife damit bekannter würde.

Erst wenn Sie die Deklaration von Antwort aus der Schleife heraus ganz an den Kopf des Eventhandlers stellen, haben Sie Lebenszeit und Zugriffsmöglichkeit der Variable für die Auswertung des Schleifen-Exits in den richtigen Rahmen gesetzt.

Sie sehen: Gar nicht so schwer. Sie werden merken: Leicht zu übersehen!

2.9.1. Und was ist mit den Scope-Popups im Inspector?

Gut bemerkt! Die werden uns beim objektorientierten Programmieren noch beschäftigen. An dieser Stelle erst einmal die Kurzerklärung, dass Sie damit die Verfügbarkeit von Programmelementen für andere Programmelemente verändern können. Nicht immer soll jede Variable und jede Methode allgemein freigegeben werden. Ganz im Gegenteil, beim objektorientierten Programmieren versucht man eher, Objekte gegeneinander abzugrenzen und nur über offizielle Wege miteinander kommunizieren zu lassen, damit man sie nicht zum Freiwild für anderen Code macht, der womöglich zu unvorhergesehenen Zuständen dieser Objekte führt.

Aber, wie gesagt, das gehört eher zum objektorientierten Programmieren, und ich will nicht zu weit vorgreifen. Das ist noch locker zwei Seiten entfernt. Genießen wir vorher noch das



2.10. **Outro**

Sie dürfen auch gerne Epilog dazu sagen. Ja sicher, wir könnten in dieser Einführung noch viel mehr in Breite und Tiefe gehen. Jede Menge! Die bisher vorgestellten Datentypen und Operatoren haben wir nur gestreift. Es gibt von beiden viel, viel mehr, und es gibt auch noch mehr über die erwähnten zu wissen.

Das wird aber, so Muße und Inspiration sich unter einem günstigen Stern begegnen, bald in Form der Fortgeschrittenen-Kapitel, der Referenzseiten und weiterer Einführungs-Kapitel folgen.

Was aber die Programmier-Grundlagen angeht, sollten Sie jetzt genügend gewappnet sein, um die Objekte mit ins Boot zu holen. Darum geht's uns ja bei Xojo, und mit halbgaren mathematischen Auswertungen habe ich Sie nun wirklich zur Genüge gequält.

Wenn Sie das alles noch einmal vertiefen wollen, hilft vielleicht auch dieses Tutorial dabei:

[Einführung in die Programmierung Teil II](#)

Ansonsten: Ist Ihr Englisch gut genug, dann werfen Sie ruhig einen Blick in die Sprachreferenz, insbesondere die Abschnitte über [Datentypen](#) und die [Operatoren](#). Dort bekommen Sie schon einen kleinen Einblick in weitere Möglichkeiten.

Und öffnen Sie ruhig mal das eine oder andere der zahllosen Beispielprogramme, die Sie direkt aus Xojo heraus finden können, wenn Sie ein neues Projekt anlegen.

Selbst wenn Sie dort nicht alles verstehen sollten: Ein bisschen Erholung haben Sie sich verdient. Seien Sie ruhig abenteuerlustig. Das eigene Interesse ist immer noch der beste Lernbooster.

Ach ja: Und wenn Sie Anregungen, Kritik, Fragen, Tipp- oder sachliche Fehler zu melden haben: [Bittebitte!](#)

Vielen Dank bis hierhin!



3

EINFÜHRUNG IN DIE OBJEKTORIENTIERTE PROGRAMMIERUNG

Hier wird weitergeführt, was in Kap. 2 vorgestellt wurde, und auf OOP übertragen.

Dazu gehören:

- OOP-Grundbegriffe: Klassen, Objekte, Instanzen, Constructors, Shared Properties & Methods und nochmals den Scope sowie ByRef, ByVal und ARC.
- Klassen: Date, Xojo.Core.Date, Xojo.Core.DateInterval
- Techniken: eigene Klasse erstellen, nutzen, optimieren
- Beispielprojekte: Zeitnehmer (Kommandozeile)



KAP. 3: Einführung in die objektorientierte Programmierung

3.1. OOP, oh weh!



Bitte anschnallen und das Knabbern vor der Tastatur einstellen – wir starten jetzt durch und erreichen in wenigen Seiten unsere Reiseflughöhe für die objektorientierte Programmierung!

Bevor wir uns am Objekt orientieren, sollten wir uns erst einmal orientieren, was so ein Objekt eigentlich ist. Wenn Sie meinen Ausführungen zum Thema binäre Struktur des Speichers auf Seite 24 gefolgt sind, dann wissen Sie, dass es darauf nur eine Antwort geben kann: Speicher! Arbeitsspeicher strenggenommen. Im sehr vergröberten Computermodell ebenda wurde es hervorgehoben: Im Grunde vermittelt die CPU nur zwischen Ein-/Ausgabegeräten (und dem Menschen dahinter) und dem aktuellen Zustand von Speicher, denn anderes kann sie gar nicht verwalten.

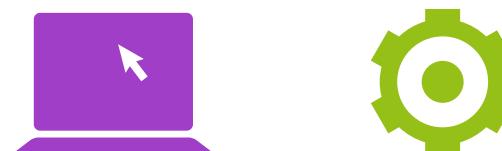


Abb. 7: Auch bei der objektorientierten Programmierung: Die CPU sieht nur Speicher-Objekte schwirren da nicht rum!

Wo liegt denn dann der Unterschied zum bisherigen prozeduralen Programmieren?

Zweierlei sind die Differenzen! Mindestens ...

- Bisher haben wir ausschließlich **Datentypen** verarbeitet: Integer, Double, Boolean, String und Text: Egal wie viele Bytes diese Datentypen jeweils belegen (bei den Zeichenketten ist das aus naheliegenden Gründen ja variabel gestaltet), eines ist ihnen gemein:

Die Bits oder zusammengefassten Bits, also Bytes oder größer, ergeben direkt den Wert eines Datentypen.

Sie können, wie auf Seite 25 gezeigt, im Prinzip (und ja: auch in der Praxis!) die Bits eines UInt8 analysieren und daraus seinen Wert erkennen. Und ebenso gilt das für alle anderen Datentypen, auch wenn deren Analyse, wie bei Text mit den variablen Zeichengrößen, komplizierter sein mag.

Es mag bei einem Datentypen noch interne Hilfsvariablen geben, etwa einen Zähler – als Integer z. B. – für die Länge einer Zeichenkette. Aber das ist schon eher die Ausnahme als die Regel. Grundsätzlich wird ein Datentyp durch Speicher einer bestimmten, auch flexiblen, Größe dargestellt, und sein Speicherinhalt definiert seinen Wert.

Und noch etwas eint sie, die Datentypen:

Ein Datentyp ist sofort nach Reservierung benutzbar – und im Falle von Xojo hat er dann auch bereits einen Vorgabewert.

Soll heißen: Wenn Sie eine Zeile wie

Dim MeinName As Text

in den Code-Editor eingeben und diese Zeile im Rahmen des Programms zur Ausführung gelangt, dann steht Ihnen ab diesem Moment – und bis MeinName aus dem Scope. gerät – die Variable mit diesem Datentyp unter dem gewünschten Namen auch zur Verfügung und hat den Vorgabewert "" – also leer.

Das gilt nicht nur für String und Text, sondern für alle Datentypen: Zahlentypen erhalten i. d. R. den Vorgabewert 0, Boolean False usw. Kalter Kaffee.

 Das ist, nebenbei gesagt, nicht bei jeder Programmiersprache so. Manche erwarten auch, dass Sie nach Variablenklärung einen Wert zuweisen, da sie keine Vorgabewerte setzen – dort könnte im Zweifelsfall also ansonsten irgendein Wert drin stehen.



Und dann wird Ihnen noch aufgefallen sein, dass Sie Datentypen normalerweise nicht umständlich entsorgen müssen. Gerät ein Datentyp aus dem Scope, bringt ihn der Compiler automatisch zum Müll – der Speicher wird wieder zur freien Nutzung gekennzeichnet.

Sie sehen, die Xojo-Küche ist echt High-Tech! Einen Müllrobo-ter könnte ich auch gut gebrauchen. Tja, und wenn die Putz- und Wischroboter schon so richtig gut wären ...

OK, nun, wenn das alles Kennzeichen von Datentypen sind – wo liegen denn die Unterschiede zu den Objekten?

Wenn Sie sich die Abbildung auf der Vorseite noch einmal anschauen, insbesondere den Speicher: Für Objekte ist da eigentlich nichts vorgesehen! Und so ist es auch. Verzeihung, falls ich Sie einer Illusion beraube, aber unsere einfache binäre Computertechnik lässt sich in dieser Hinsicht eigentlich nicht tunen. Wenn wir gemein sein wollen, können wir mit Fug und Recht behaupten:

Der Computer kennt keine Objekte!

Sollte Ihnen ob dieser frechen Behauptung ein homerisches D'oh! entgleiten, rudere ich mal lieber schnell zurück: Er kennt ja eigentlich auch keine Texte, keine Dezimalzahlen ... nur seine Bits und Bytes! Dass er diese unterschiedlich interpretiert, liegt an unserer Übereinkunft mit ihm.

Dass wir seine Phantasie soweit beanspruchen können, alsdass er nun Speicherabschnitte wie Objekte betrachtet, das liegt daran, dass sich eine ganze Batterie von Methoden und Funktionen vermittelnd zwischen CPU und RAM drängelt. Und damit Ihre Kladde weiter wachsen kann:

Eine Sammlung von Methoden, Funktionen, Klassen und weiteren Programmelementen, die gemeinsam einen bestimmten Programmier- oder Betriebssystemaspekt unterstützen, bezeichnet man auch als

Framework.

So hatte ich ja bereits das neue oder auch Xojo-Framework erwähnt. Nun, darin tummelt sich alles, was die moderneren Versionen einiger Datentypen und Klassen beinhaltet, und es ist stringenter punktnotiert.

Ein anderes Framework für Apple-Programmierer wäre [Sprite-Kit](#), das für die schnelle und effektive Programmierung von 2D-Spielen und -Physiksimulationen auf iOS und macOS existiert.

Das wird Sie jetzt also nicht überraschen: Auf Betriebssystemebene schaltet sich, wenn es um das Simulieren von Objektstrukturen geht, ein Objekt-Framework ein, englisch also ein Objective Framework, und weil die Systeme zumeist in C programmiert sind, heißt das Ding auf macOS und Linux dann auch

Objective-C Framework

Unter [Windows](#) ist das nicht so fein unterteilt, aber es finden sich natürlich ähnliche Klassen – wie ich auch sonst hier nicht groß auf Unterschiede eingehen. Die Namen mögen verschieden sein, aber immer gilt: Es gibt ein oder mehrere Frameworks, die aus Speicherbereichen Objekte simulieren und sie so fürs System (und den Programmierer) handhabbar machen.



Abb. 9: Frameworks machen's möglich: Das Objective-Framework verwaltet Speicherbereiche als Objekte, das Grafik-Framework bringt sie auf den Bildschirm.

So sind die Frameworks also Funktionsbibliotheken. Und da Xojo eine objektorientierte Sprache ist, werden wir uns in erster Linie mit Frameworks und Funktionen beschäftigen, die auf diesem Objekt-Framework aufsetzen.

Und da Xojo auf vielen

nativen,

also den „echten“, im Betriebssystem der jeweiligen Plattform, vorhandenen Objekten aufsetzt, können Sie diesen Abschnitt auch als kleine Einführung in die Maschinerie begreifen, die unterhalb von Xojo werkelt.

Grundsätzlich also ist ein Objekt auch erst einmal nur Speicher. Wieviel davon pro Objekt, das muss uns nur in den seltensten Fällen Gedanken bescheren. Da kümmert sich das entsprechende Framework drum. Der große Unterschied zum Datentyp ist aber:

Die Bits eines Objekts geben nicht zwangsläufig direkte Auskunft über seinen Wert!

Was zum einen daran liegt, dass ein Objekt nicht zwangsläufig EINEN Wert besitzt.



Ein Fenster zum Beispiel bei einem Desktop-Projekt, also eine spezialisierte Unterart eines Objekts innerhalb eines objekt-orientierten Betriebssystems, besitzt jede Menge „Werte“ – den Abstand zum Desktop beispielsweise, seine Breite und Höhe, seinen Titel, seine Hintergrundfarbe, andere Objekte, die auf ihm platziert sind – da kommt was zusammen!

Da es bei dieser Vielzahl unpassend wäre, vom „Wert“ des Fensters zu sprechen – welcher soll das sein? –, macht man das auch nicht. Statt Werten also besitzen Objekte

Eigenschaften.

Im Englischen Properties, und da dieser Begriff auch innerhalb der IDE verwendet wird, werde ich ihn in der Regel unübersetzt benutzen. Falls Sie

Property = Eigenschaft

in diesem Zusammenhang noch nie gehört haben: Das wäre mal ein Fall fürs Auswendiglernen: Property, Property, Property – den Begriff werden Sie zum Ausgleich jetzt andauernd hören!

Was genau ist eine Property? Zum Verständnis dieser Frage ertragen Sie bitte kurz meinen

3.1.1. Rant über die Laxheit im Umgang mit der Sprache

Wäre es nicht prima, wenn all diese Seiten überflüssig wären und Xojo sich einfach per Sprachkommando programmieren ließe? Warum ist der Computer nicht so weit? Sogar auf der alten Enterprise ging das doch schon!

Ich denke, er wäre schon längst dazu in der Lage und weitaus besser als [Siri](#) das heute und auch morgen können wird (für übermorgen geb ich mal lieber keine Prognose ab), wenn wir uns nur mehr Mühe gäben!

Im alltäglichen Sprachgebrauch sagen wir mal „der ist laut“ oder „heut isser aber blau!“ und meinen womöglich, damit eine Aussage zu einer Eigenschaft einer anderen Person abgegeben zu haben. Daraufhin setze ich aber mein Klugscheißerkäppchen auf und doziere: „Hamwa aba nich!“

Bestenfalls eine Vermutung wurde abgefeuert, denn welcher Eigenschaft soll der Ungefähr-mal-so-Wert „laut“ zugeordnet werden? Sprachlautstärke? Körperfunktionen? Bewegungsgeräusche? Und ist blau der Wert der Eigenschaft Körperfarbe (ich hoffe nicht!)? Der Alkoholisierungsgrad? Geht's nicht genauer?

Also, eben nur um unseren Sprachgebrauch mal abzustrafen und uns computerverständlicher zu machen: Eine Property kann einen Wert annehmen (und dieser kann durchaus ein normaler Datentyp sein – siehe die Size-Felder im Bild rechts –, aber auch ein Objekt, eine Klasse, eine Aufzählung und einiges mehr. (Keine Angst, das klärt sich bald!) Eine Property ist jedenfalls nicht ihr Wert – sie hat einen (oder auch mal keinen)!

Beeep beeep bop bop ...

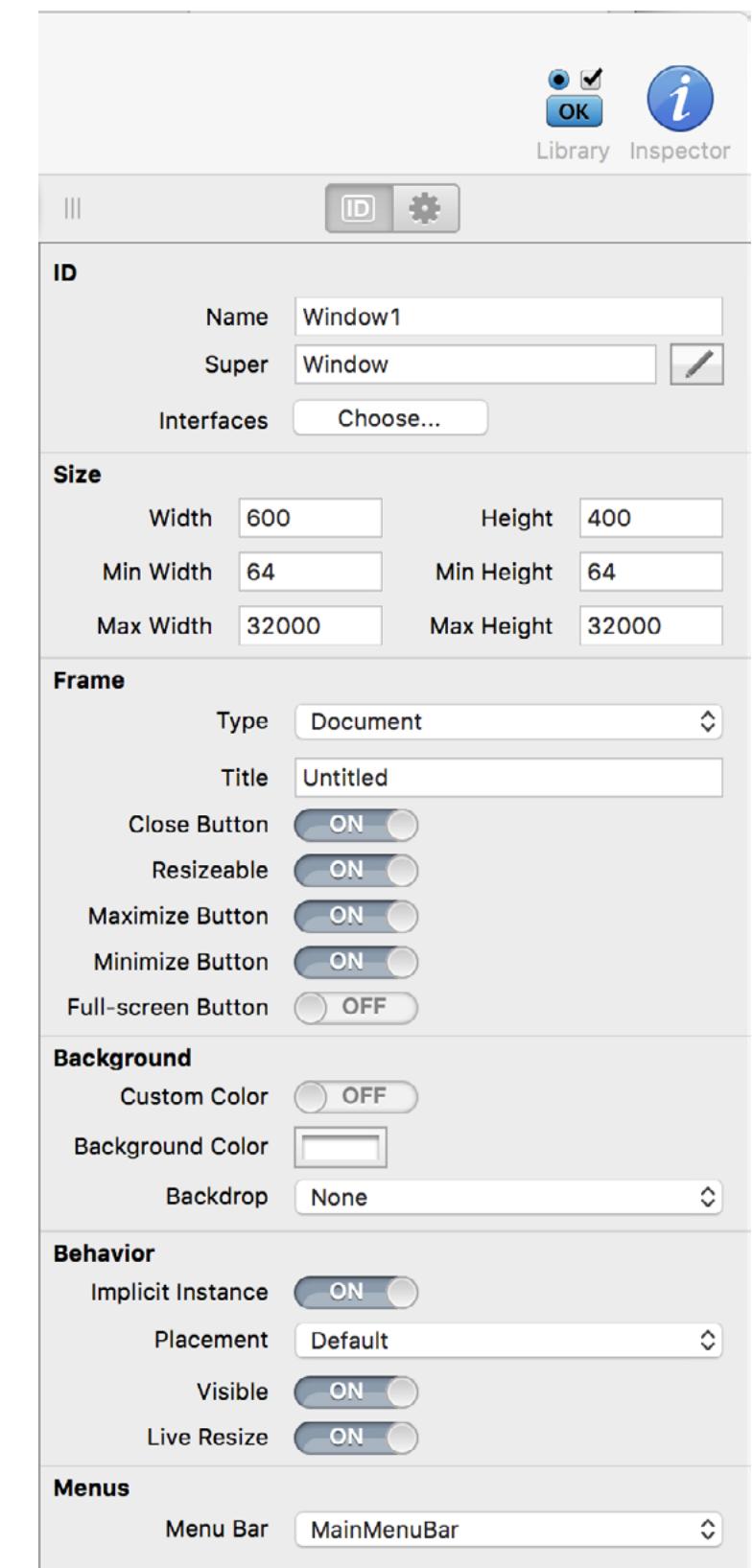


Abb. 8: Nur mal so zum Gucken: einige Fenster-Properties



3.1.2. Klassenbewusstsein

Was in einer zivilisierten Gesellschaft nun wirklich mal abgeschafft gehörte – das Denken in Kassen und Klassen –, ist beim Computer andererseits absolut hilfreich.

Wenn Sie sich das Bild mit (nur einigen!) der Properties eines Fensters – oder um auch hier wieder zur Originalsprache zurückzukehren, wie Sie sie ja auch in der IDE vorfinden: eines Windows – auf der Vorseite noch mal anschauen, leuchtet eines schnell ein: Ein Computer braucht, soll er ein Fenster erzeugen, irgendwoher eine Vorlage dafür! Er braucht den Bauplan, um zu wissen, welche Properties er anlegen soll und an welcher Stelle (und noch so einiges mehr!).

Diese Baupläne existieren, und zwar wieder mal in den verschiedensten Frameworks. Und sie heißen

Klassen

bzw. Englisch, Singular:

Class

Bei der fast buchstäblichen Identität beider Begriffe fällt es mir diesmal schwer, konsequent den englischen zu benutzen. Ich bleibe konsequent willkürlich und folge dem Sprachgefühl – suchen Sie im Index eher nach dem deutschen Begriff!

Ein Bauplan ist aber nicht das Ding an sich. Andernfalls hoffe ich, dass Ihre 400.000 € teure Architektenzeichnung Ihnen genug Trockenheit und Wärme spendet.

Um einen Bautrupp herbeizurufen, der mithilfe des Bauplans und nötiger Materialien (die in der Regel aus genügend viel nutzbarem, da freiem Speicher bestehen) den Rohbau eines Objekts zimmert, bemühen wir in Xojo einen alten Bekannten:

Dim

Dim, dimensioniere, reserviert Speicher. So habe ich den Befehl seinerzeit vorgestellt. Das gilt nicht nur für Datentypen, das gütet genauso sehr für Objekte.

 **Obacht!** Bewusst spreche ich vom **Rohbau** des Objekts! Dim reserviert nur den Speicher, er ist damit **nicht bezugsfertig!**

Das hat seine Gründe: Manchmal will man gar kein neues Objekt erzeugen. Es genügt, irgendeine Funktion aufzurufen, die ein Objekt dieser Art – einer solchen Klasse also – als Ergebnis liefert. Entsprechend benötigt man nur den Speicher für das Objekt, und den Innenausbau erledigt dann die andere Funktion. Ein Beispiel:

Die Klasse

Xojo.Core.Date

ist die Klasse des neuen Xojo-Frameworks, mit der sich Daten verwalten und vergleichen lassen – im Sinne von Kalenderdaten. Ob Sie die Nanosekunden vergleichen wollen, die zwei Funktionsaufrufe benötigen, oder ob Sie einen Zeitstrahl über

ein paar Jahrhunderte aus einer Datenbank heraus aufbauen wollen – Funktionen dafür finden Sie in dieser Klasse.

Häufig, z. B. gerade bei Zeitintervallmessungen, benötigt man die aktuelle Zeit. Dazu existiert die Methode

Xojo.Core.Date.Now,

die das aktuelle Datum in Form eines Objekts der Klasse Xojo.Core.Date zurückliefert. Auf die Nanosekunde. Will ich die Zeit nehmen, hilft dieser Code:

```
Dim Jetzt As Xojo.Core.Date = Xojo.Core.Date.Now
```

 Das können Sie natürlich auch in zwei Programmzeilen erledigen – erst Dim, dann zuweisen –, aber das wissen Sie ja. Wollt's nur noch ein letztes Mal gesagt haben!

Tja, und manch anderes Mal, da will man eben ein neues Objekt erzeugen. Da reicht dann kein Rohbau, es muss durch das eigene Team bezugsfertig gemacht werden. Der hat zum Glück eine kurze Telefonnummer: unter

New

erreichen Sie ihn.

New initialisiert ein neu reserviertes Objekt. Technisch gesehen werden einige Referenzen zur Mutterklasse gesetzt, Startwerte der Properties gesetzt und was immer dem Ersteller der Klasse sonst noch eingefallen ist.



Eine bezugsfertige neue Instanz einer Klasse erhalten Sie also durch

```
Dim MeinObjekt As MeineKlasse
MeinObjekt = New MeineKlasse
```

oder eleganter durch

```
Dim MeinObjekt As New MeineKlasse
```

Dieses Abarbeiten eines Bauplans – einer Klasse – bis hin zur Bezugsfertigkeit hat auch einen Fachbegriff:

Instanziieren.

Oder anders ausgedrückt:

```
Dim MeinObjekt As New Klasse
```

erzeugt eine Instanz der Klasse Klasse,

die genau wie von Datentyp-Variablen gewohnt unter ihrem Namen – also **MeinObjekt** – verwendet werden kann.

 **Aufgemerkt!** Für den Fall, dass Sie das Beispiel am lebenden Objekt ausprobieren und Ihnen eine Fehlermeldung entgegenschlägt, wenn Sie New mit Xojo.Core.Date testen: **Die genaue Funktionsweise von New ist klassen-abhängig.** Manche Klassen erwarten Initialisierungsparameter – so auch Xojo.Core.Date. Ich will aber nicht zu weit vorgreifen ...

Also, um's noch einmal festzuhalten (Sie ahnen es schon längst: Auch hier wieder ein echter Bug-Hit!):

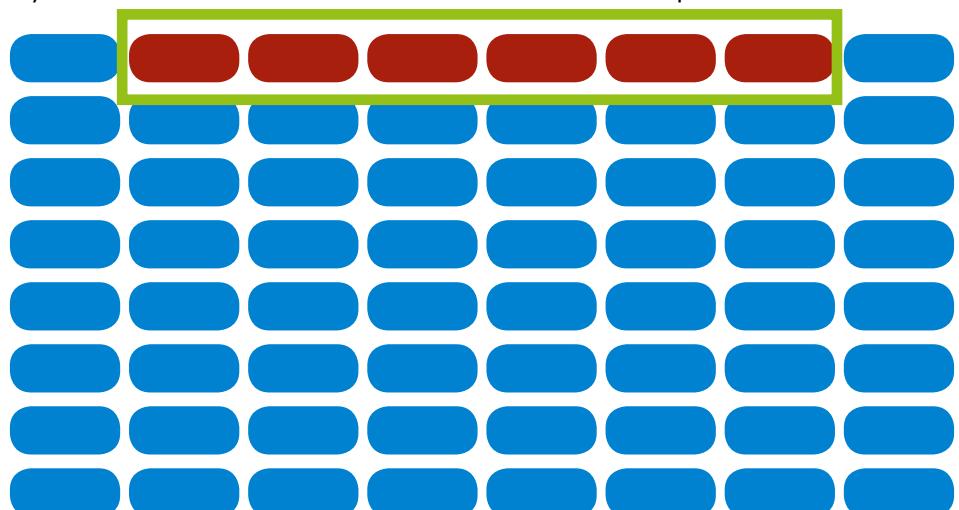
Dim reserviert den Speicher für eine Instanz einer Klasse



Dim MeinObjekt As MeineKlasse

reserviert den Speicher für ein Objekt der Klasse **MeineKlasse**, die Bestandteil eines Frameworks ist:

Bytes im RAM; rot = für Instanz reservierter Speicher.



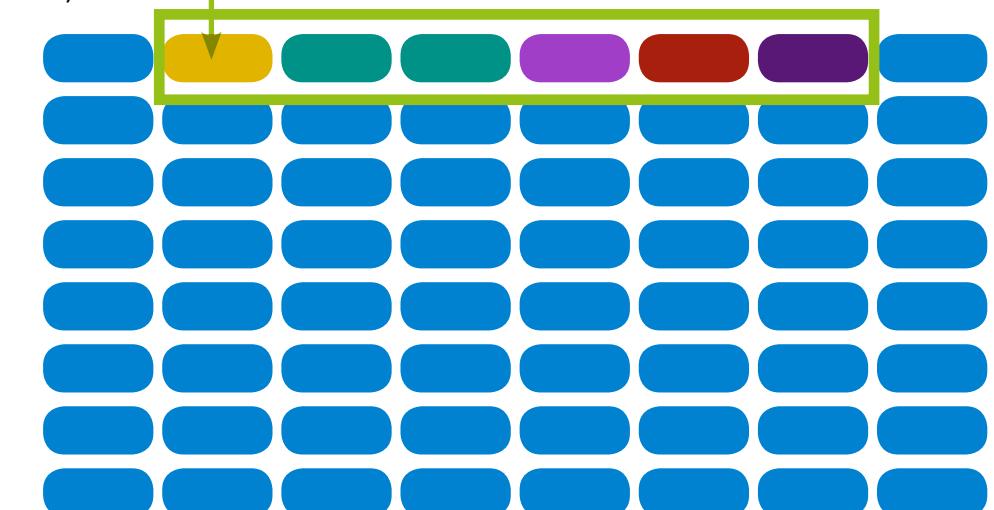
New initialisiert den reservierten Objektspeicher



MeinObjekt = New MeineKlasse

setzt Querverweise, initialisiert Properties mit ihren benutzerdefinierten oder Vorgabestartwerten

Bytes im RAM; alles außer Blau: definierte Werte



Wenn dies so ein beliebter Treffpunkt für sechsbeinige Krabbelgruppen ist: Was passiert, wenn man versucht, ein nicht-initialisiertes Objekt zu nutzen?



Ausnahmsweise bemühe ich hier doch einmal das alte Framework, obwohl es eine neue Entsprechung gibt. Aber wenigstens habe ich diese schon vorgestellt: `Xojo.Core.Date`. Die ältere Klasse hört einfach auf den Namen

Date.

Ich greife jetzt zum Altobjekt, weil `Date` sich mit einfacherem `New` initialisieren lässt, im Gegensatz zum neuen, das wie gesagt Initialisierungsparameter haben will.

Ein

```
Dim d As New Date
```

erzeugt überall – außer bei iOS, wo es das alte Framework nicht gibt! – ein initialisiertes `Date`-Objekt mit Namen `d` – initialisiert auf den Zeitpunkt der Erstellung,

`New (Date)` ist also die Entsprechung zu
`Xojo.Core.Date.Now`

Sehnse, sehnse! Wegen dieser Umständlichkeiten will ich mich lieber auf das neue Framework konzentrieren und das alte nur der Vollständigkeit halber referenzieren. Ich versuche, Ausnahmen dieser Art selten bleiben zu lassen!

► Werfen Sie jetzt bitte Xojo an (oder kehren Sie zu Ihm zurück) und erstellen Sie ein neues Console-Projekt. Sie wissen ja, die das geht. Falls nicht mehr – Seite 29!



Öch was, schon wieder die Kommandozeile? Ich dachte, wir programmieren objektorientiert!?

Falls Sie gerade Ähnliches gedacht haben sollten, war's ja gut und ganz persönlich für Sie gedacht! Objektorientiertheit wird gerne mit graphischen Benutzeroberflächen in einen Topf geworfen, vielleicht, weil die Steuerelemente – die Buttons, Textfelder, Tabellen, Fenster ... – ihrerseits objektorientierten Betriebssystemen erwachsen und sie darin durchaus Objekte darstellen. Allerdings eine Untermenge, als visuell sichtbare

Steuerelemente

oder auch

Controls,

wobei es durchaus auch nicht-sichtbare Controls gibt.

Fazit also:

Ja, Controls entstammen heute meistens objektorientierten Frameworks. Nein, Objektorientiertheit heißt nicht, dass graphische Steuerelemente vorhanden sein müssen.

Und das App-Objekt, dessen Run-Eventhandler wir füllen: Rein gelegt! Wir haben von Anfang an objektorientiert programmiert. Mehr oder weniger zumindest!

► Fügen Sie dem App-Objekt einen Run-Eventhandler hinzu.

3.1.3. Eventhandler hinzufügen

Eine Methode habe ich Ihnen bereits auf Seite 30 erklärt. Wenn Sie weitere Möglichkeiten kennenlernen möchten: Schauen Sie mal auf Seite 43. Was dort über Methoden geschrieben steht, gilt wortwörtlich auch für Eventhandler. Außer, dass Sie natürlich nach dem Begriff „Event handler“ fahnden müssen statt nach „Method“ ...

► Füllen Sie den Code-Editor des Eventhandlers hiermit:

```
Dim d As Date // Kein New!!! Absicht!!!
d.Day = 18
```

Day

ist eine Property der Date-Klasse. Sie sehen daran den Tag des Monats als Integer und können ihn auch setzen. In diesem Beispiel also würde das Datum auf den 18. gesetzt werden.

► Lassen Sie das Projekt laufen.

Das Programm startet, doch nach kurzer Bedenkpause haben Sie nun Gelegenheit, Kontakt zu einem weiteren, sehr wichtigen Bestandteil der Xojo-IDE aufzunehmen. Nehmen Sie sich Zeit dazu und freunden Sie sich gut an. Den Knaben hier werden Sie noch oft zu sehen bekommen: Es ist



3.2. Der Debugger

Auf den ersten Blick sieht das nicht so viel anders aus als die Fehlermeldung von Seite 33, oder? Wenn Sie genauer hinschauen schon:

- zum einen sind da jetzt Schaltflächen, wo sonst der Editor seine Positionierungswerzeuge anzeigt. Die ersten beiden sehen nach Start und Stopp aus, und das sind sie auch:

Der Debugger gestattet Ihnen einen Live-Einblick in das laufende Programm. Es wird dazu angehalten und die aktuelle Code-Position wird angezeigt. **Das Käfersymbol** an der letzten Zeile zeigt uns, dass an dieser Stelle ein Fehler offenbar wurde.

Es bedeutet nicht, dass der Fehler an genau dieser Stelle sitzt!

Diesmal wurde die Fehlermeldung nicht vom Compiler erzeugt, der vor Fertigstellung des Programms einen Tipp- oder Syntaxfehler gefunden hat. Soweit war alles in Ordnung, nur nicht langfristig lauffähig: Das Programm hat einen Bug!

- und zum anderen, und das bringt uns dem eigentlichen Fehler auf die Spur, sehen Sie im Panel rechts unten unter „Variables“ ebendiese gerade vorhandenen mit ihren aktuellen Werten. Lassen Sie sich von der Anzahl nicht irritieren: d hat den Wert **Nil**, und darunter gibt es eine Variable **Exception** mit dem Wert **NilObjectException**. Wenn die mal nichts miteinander zu tun haben ...!

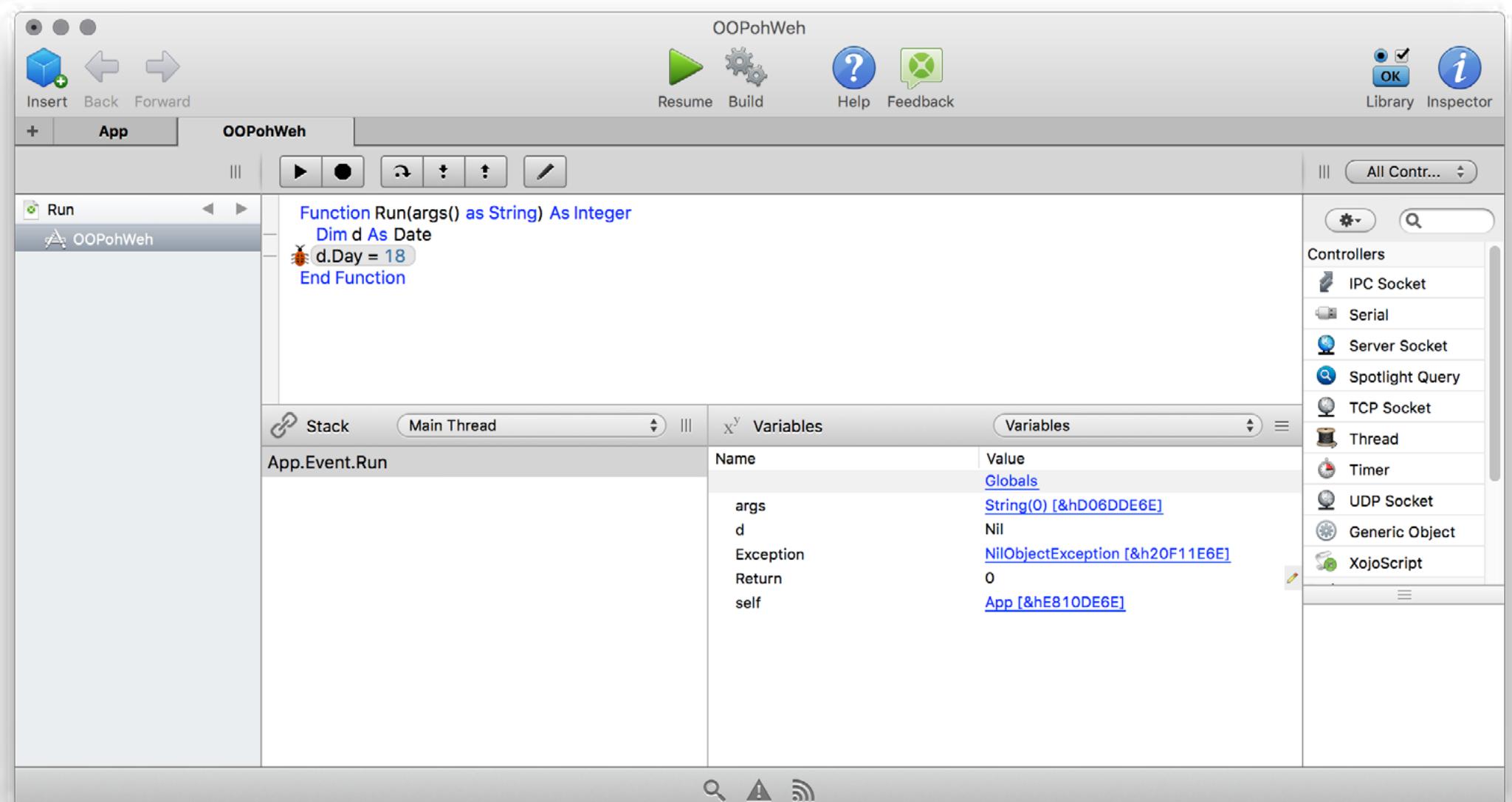


Abb. 10: Nenn ihn lieben Vertrauten, nicht „Oh nein!“ – Der Debugger ist Programmierers bester Freund!



Haben Sie natürlich und sollen hier auch Erklärung finden:

Nil

ist der Computerslang für *Nicht vorhanden, zeigt ins Nirvana ...*

Ein Objekt, das Sie benutzen wollen, darf niemals **Nil** sein. Das ist das Zeichen dafür, dass Sie nur einen Rohbau vor sich haben: Einen reservierten Speicherbereich, der aber nicht standesgemäß initialisiert wurde.

Entweder haben Sie hier ein **New** (eventuell mit entsprechenden Startparametern) vergessen, oder Sie wollten sich eigentlich einen Objektwert von irgendeiner Funktion liefern lassen, versäumten aber, bei ihr vorstellig zu werden, oder aber, was auch vorkommen kann, die Funktion hatte gerade keine Meinung zu Ihrer Frage. Dann liefern viele Funktionen, die eigentlich ein Objekt hervorbringen sollten, den Wert **Nil**. Man könnte also sagen:

Nil ist der Vorgabewert für Objekte – solange sie nicht initialisiert sind.

Zeit für eine weitere meiner beliebten Konditionalkonstruktionen:

Wenn **Nil** ein Standardwert für nichtvorhandene Objekte ist, dann muss man Objekte auch daraufhin prüfen können.

Und natürlich kann man das! Ein

If d <> Nil Then ...

vor der Zuweisung des Tageswerts hilft, die Exception zu vermeiden.

Gutes Stichwort, oder? Was ist eigentlich die hier aufgetauchte

Exception?

Vermutlich ganze Generationen von Computeranwendern wurden traumatisiert vom Auftauchen eines blauen Bildschirms, verbunden mit dem Worten

Ein schwerwiegender Ausnahmefehler ist aufgetreten!

Ich habe nicht wenige Windows-Benutzer gesehen, die sich schuldbewusst wegduckten, denn irgendwer muss den Fehler ja gemacht haben – also wer, wenn nicht der Anwenderlaie?

Auf dem Mac wurde früher an dieser Stelle mit Bomben um sich geworfen. Heieiei, wenn Apple dafür mal nicht noch fetter auf der NSA-Liste landet!

Die auslösende Situation war auf beiden Rechnern dieselbe und nur selten dem Anwender in die Schuhe zu schieben:

Das System hatte bemerkt, dass dort, wo eigentlich ein definierter, sicherer Wert sein sollte (bzw. eben eher ein Objekt, das durch irgendeines seiner Properties zu diesem Wert beitrug), keiner vorhanden war. Wenn dieser Umstand nicht durch eine Abfrage wie nebenstehende ausgegrenzt wurde, sondern der Programmcode ein fröhliches Weiterrechnen vorgab, tja, dann wusste sich das System keine andere Lösung als den Ausnahmestatus zu verhängen!

Das ist also das Außergewöhnliche an der Exception: Es ist ein Zustand eingetreten, der ein Weiterfunktionieren des Programms nicht mehr garantieren kann. Was soll beim Rechnen mit Nichts für ein Ergebnis herauskommen? Da kann man ja gleich durch Null teilen! Wo kämen wir da hin?

Vielleicht, sollten Sie zu den Blue Screen-Traumaopfern gehören, verschafft es Ihnen nachträglich Linderung, wenn ich Ihnen daher sage:

Ein Ausnahmefehler hat nur selten etwas mit dem Benutzer oder mit einem Defekt des Rechners zu tun. Viel häufiger hat der Programmierer versäumt, einen Zustand sicherzustellen, der berechenbare Daten für sein Programm garantiert.

Eine

NilObjectException,

um das noch vollständig abzuschließen, ist eben ein möglicher Fall eines solchen Ausnahmezustands, und deutet darauf hin, dass im Programm irgendwo ein Objekt existiert, das in Wirklichkeit keines ist – also den Wert **Nil** hat –, obwohl es eines sein sollte. Eine **KeinObjektAusnahme**. Klingt nach Til Schweiger ...

► Entsprechend gibt es noch eine Vielzahl weiterer Exceptions, die wir uns später noch genauer anschauen. Ebenso wie die weiteren Möglichkeiten des Debuggers. Und glauben Sie mir: Gelegenheit zum besseren Kennenlernen werden Sie zuhauf haben! Mit beiden!

Ein Debugger-Feature allerdings möchte ich schon jetzt verraten.



- ▶ Beenden Sie das noch laufende, aber fehlerhafte Programm, indem Sie
 - auf das Pfeilsymbol am Kopf des Debuggers drücken, woraufhin es sich durch die Exception beendet
 - auf den dicken Punkt daneben klicken, um das Programm via Debugger zu beenden
 - den neuen Tab in der IDE, der den Namen Ihres Projekts trägt und den Debugger beinhaltet, über das Schließfeld des Tabs schließen oder
 - das Terminalprogramm Ihres Rechners beenden.
- ▶ Verändern Sie den Run-Eventhandlers zu diesem Code:

```
Dim d As New Date
d.Day = 18
Break
```

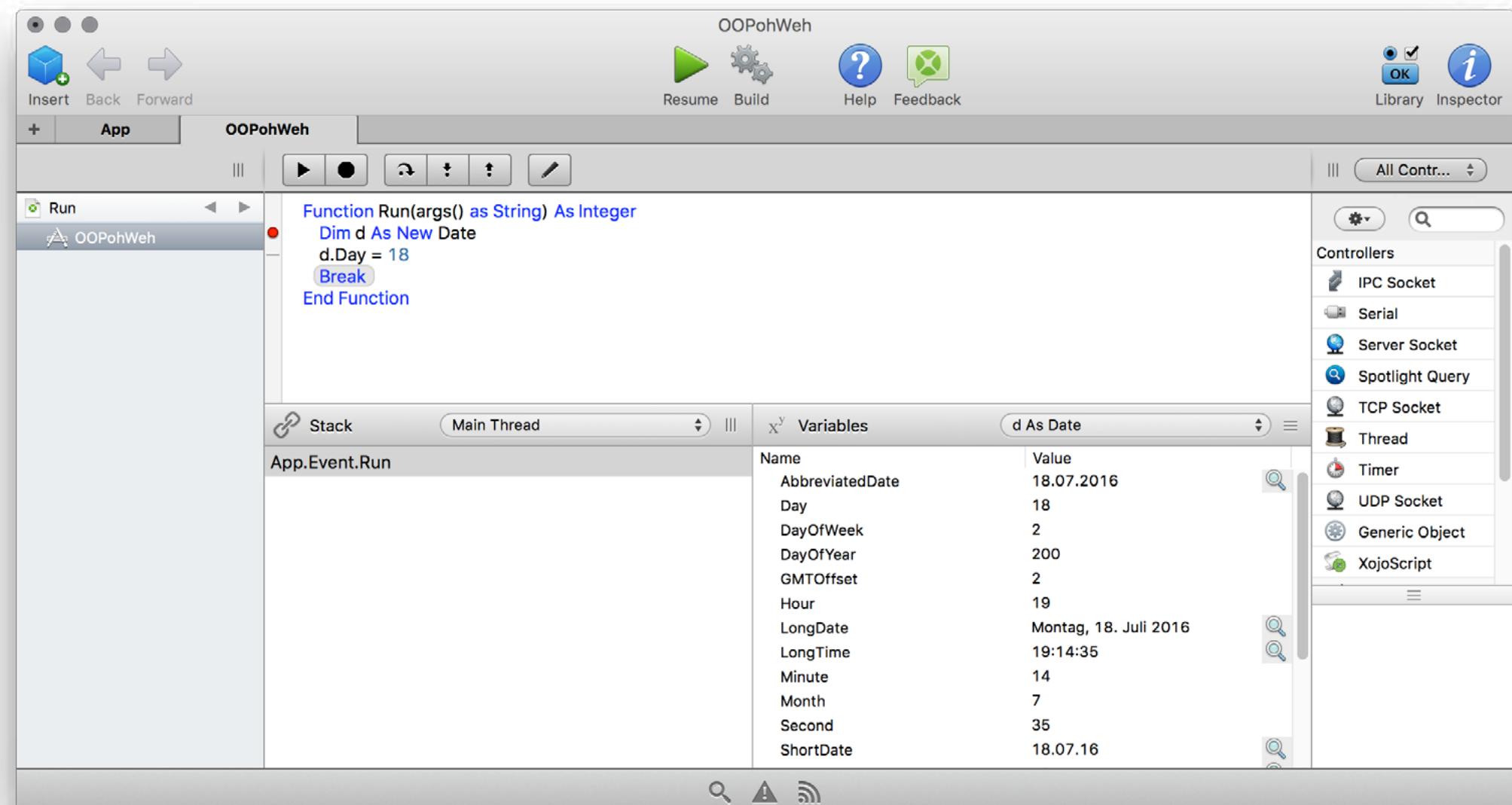
- ▶ Starten Sie das Projekt.

Es wird nun ganz normal ablaufen, da d mit einem New initialisiert wird. Der Befehl

Break

aber ruft absichtlich den Debugger hervor und pausiert das Programm. Sie sehen: Keine Käfer in Sicht, d ist nicht `Nil` (was es stattdessen ist, folgt in Kürze), und Sie können sogar in bester Browser-Manier auf das blau unterstrichene d klicken, um sich seine Properties anzuschauen.

 Das ist wieder mal ein echtes Knoten-Ins-Taschentuch-Mach-Feature! Sie werden es oft benötigen, um Fehlern auf die Spur zu kommen oder um schnell mal die Richtigkeit einer Programmrechnung zu prüfen.



Eine zweite Möglichkeit zum Schnell-Debuggen habe ich auch angeskizziert: Sehen Sie den roten Punkt an der `Dim d ...`-Zeile? Den setzen Sie, indem Sie an den Beginn einer Codezeile, dort, wo sonst ein horizontaler Strich sitzt, klicken.

Er hört auf den Namen

Breakpoint

und macht ganz ähnliches wie der Break-Befehl: Er hält die Programmausführung an und katapultiert das Programm in den Debugger, wenn es an einer solchen Codezeile ankommt.

Der Unterschied ist, dass ein Breakpoint ohne Codeänderung gesetzt werden und auch schnell wieder ausgeschaltet werden kann – sowie alle gleichzeitig per Menübefehl im

Project-Menü/Breakpoint/Clear All



3.3. Auf die Plätze ...

Ich weiß nicht, wie Sie das sehen, aber ich finde, wir haben uns ein bisschen Praxis verdient. Ich, der ich seinerzeit mit ausschließlich objektorientiertem Programmierwissen zur Xojo-Expedition aufbrauch, habe Wochen, wenn nicht Monate benötigt, um mein rein prozedurales Wissen auf OOP upzudaten.

Sollte Ihnen der Kopf brummen: Das ist ziemlich normal! Falls nicht: Schreiben Sie mir! Würde mich freuen, hätten Schweiß und Tränen der Vergangenheit einen nachvollziehbaren Weg aufgedeckt ...

Jetzt, da wir wissen, dass ein neues Objekt initialisiert werden muss und es mehrere Properties beinhalten kann, schauen wir uns ein schon kurz erwähntes genauer an und nutzen seine Funktionen. Und da ich Ihnen bevorzugt die neuen Framework-Features erklären und zu ihrer Benutzung animieren möchte, übertünchen wir den kurzen Ausflug zum Date-Objekt und widmen uns wieder Xojo.Core.Date.

Ich erwähnte ja schon, dass Sie's vortrefflich zum Timen auch sehr kurzer Zeitabstände benutzen können. Nun, ich kenne Ihre Absichten nicht, aber zu meinen gehört, neben Programmieren auch effektives Programmieren zu erläutern. Das bedeutet manches Mal, sich mit seinem eigenen Code auseinanderzusetzen, tja, und hin und wieder auch mal Zeitmessungen vorzunehmen.

Da gibt es zwar auch eingebaute Möglichkeiten – aber selbstgemacht ist individueller! Ich möchte also:

- ▶ Ein neues Xojo.Core.Date-Objekt mit dem Wert von Xojo.Core.Date.Now anlegen,
- ▶ irgendeine Funktion aufrufen, deren Zeit ich nehmen will und dann
- ▶ ein weiteres Xojo.Core.Date.Now-Objekt erzeugen sowie
- ▶ die vergangene Zeit zwischen beiden auswerten.

Klingt nicht wild, oder? Ist es auch nicht! Legen wir mal los:

- ▶ Erstellen Sie in Xojo ein neues Console-Projekt (jaja, ich weiß!)
- ▶ Legen Sie einen neuen Run-Eventhandler in seinem App-Objekt an und füllen Sie dessen Code mit:

```
Dim StartZeit As Xojo.Core.Date = ¬  
    Xojo.Core.Date.Now  
Dim TestText As Text = "Ich will einmal ¬  
    wissen, wie lange Xojo benötigt, um eine ¬  
    Text-Variablen für diesen Text zu erzeugen!"  
Dim EndZeit As Xojo.Core.Date = ¬  
    Xojo.Core.Date.Now  
Dim Differenz As Xojo.Core.DateInterval = ¬  
    EndZeit - StartZeit  
Print Differenz.NanoSeconds.ToText
```

- ▶ Lassen Sie das Projekt laufen.

Es dauert nicht lange, und sie sehen eine Zahl vor sich. So viele Nanosekunden sind zwischen den Erstellungszeitpunkten der beiden Date-Objekte vergangen.

Es gibt hier eine neue Klasse, nämlich Xojo.Core.DateInterval.

Während Xojo.Core.Date ein exaktes Datum, also einen **Zeipunkt** darstellt, dient die DateInterval-Klasse dem Verwalten von **Zeiträumen**. Und sie kann zum Rechnen mit Date-Objekten des Xojo-Frameworks benutzt werden:

EndZeit - StartZeit ergibt im Code ein DateInterval, das der Differenz zwischen beiden Zeitpunkten entspricht.

Das DateInterval selbst besitzt eine kleine Palette an Properties:

Years, Months, Days, Hours, Minutes, Seconds,
Nanoseconds

Allesamt Integers und in ihren Werten zusammengezählt die Gesamtdifferenz ergebend, nicht jede für sich – ein Xojo.Core.DateInterval über eine Minute hat eine Minutes-Property mit dem Wert 1 und eine Seconds-Property mit dem Wert 0, nicht 60.

Es reicht in diesem Fall völlig, die Nanoseconds auszuwerten – Reservieren und Zuweisung einer einfachen Zeichenkette benötigen nie im Leben eine Sekunde oder mehr.

Genauso plausibel, wie die Differenz zwischen zwei Daten ein Intervall ergibt, funktionieren auch sonstige Rechnungen:



```
Dim NeuesDatum As Xojo.Core.Date = ¬  
    EndZeit + Differenz
```

errechnet im Beispiel ein neues Datums-Objekt, das um Differenz weiter in Richtung Zukunft liegt als EndZeit – sofern Differenz insgesamt positiv ist, was nicht unbedingt sein muss.

Das Gegenteil gilt auch:

```
Dim NeuesDatum As Xojo.Core.Date = ¬  
    EndZeit - Differenz
```

und ab geht's Richtung Vergangenheit (bei einem positiven Differenz-Wert)

3.3.1. Schaffen wir das?

Falls Sie, was ja passieren mag, eine bestimmte Zeitdifferenz selbst erzeugen wollen? Mit dieser Frage kommen wir zu einer weiteren Besonderheit von Objekten, die aber eigentlich schon angesprochen wurde:

Die allermeisten Klassen besitzen einen Constructor

Stöhnen Sie nicht und denken, die Vokabeln würden auch nie aufhören! Na gut, tun sie auch nicht. Hier versteckt sich aber nur ein alter Bekannter unter neuem Namen. Der Constructor ist die Methode, die beim Instanziieren eines Objekts mittels New aufgerufen wird. Die Aufbaumethode. Also die Funktion, die aus dem Rohbau ein benutzbare Objekt macht.

Viele Klassen besitzen einen Constructor, der ohne Parameter aufgerufen wird – ein einfaches New MeineKlasse reicht dann.

Andere Klassen hätten gerne mehr Input für den gelungenen Start, und gar nicht so selten gibt es auch Klassen mit mehr als einem Constructor, um verschiedene Arten der Initialisierung zu ermöglichen.

Im Zweifelsfall hilft immer nur der Blick in die Sprachreferenz. Beim [DateInterval](#) haben wir Glück: Es gibt einen einzigen Constructor, und er hört auf

Constructor (Years As Integer = 0, ¬
 Months As Integer = 0, Days As Integer = 0, ¬
 Hours As Integer = 0, Minutes As Integer = 0, ¬
 Seconds As Integer = 0, ¬
 Nanoseconds As Integer = 0)

Offenbar also müssen wir die Properties für das gewünschte Zeitintervall in absteigender Größe übergeben, beginnend beim Jahren. Oder müssen wir nicht? Was machen die ganzen = 0 in den Übergabeparametern?

Auch schon wieder fast zu viel Erklärung, weil der Code für sich spricht, oder? Die Zuweisungen werden dann aktiv, wenn Sie keinen Parameter übergeben – sie machen die Variablen zu optionalen Parametern. Beispiel:

```
Dim I As New Xojo.Core.DateInterval (1, 2, 3)
```

erzeugt ein Interval mit der Dauer 1 Jahr, 2 Monate und 3 Tage. Die restlichen Properties werden automatisch auf 0 gesetzt.

Wollen Sie nur einen Tag zurück, müssen Sie die übersprungenen Zeitgrößen natürlich mit angeben:

```
Dim I As New Xojo.Core.DateInterval (0, 0, -1)
```

wäre das dann.

Oder alternativ, da der Constructor im Zweifelsfall ja jede Property mit 0 auffüllt:

```
Dim I As New Xojo.Core.DateInterval  
I.Days = -1
```

Sie können ja mal testen, was schneller geht!

Wieder mal eine kleine Anmerkung für **Fremdsprachkundige**: Einige Sprachen erwarten bei Methodenaufruf grundsätzlich eine Klammer, auch wenn gar keine Parameter übergeben werden. Wenn Sie wollen, können Sie das auch schreiben, also
`Dim I As New Xojo.Core.DateInterval()`
Sie müssen aber nicht.

Um diesen Umstand noch einmal ganz ausführlich und auf Farbe zu (be)schreiben:

Der Constructor ist die Methode, die bei einem New zur Instanzierung eines Klassenobjekts durchlaufen wird.

Versuchen Sie also nicht, einen Constructor mit Constructor (Parameter) o.ä. aufzurufen – der Schlüssel ist New Klasse (Parameter)!



3.4. Auf Montage

Das ist übrigens alles, was es zum `Xojo.Core.DateInterval` zu sagen gibt. Es ist eine sehr überschaubare Klasse, die nur ein paar Integer-Properties beherbergt.

Damit wird es aber höchste Zeit, genauer in die Struktur von Klassen hineinzuschauen, oder? Eine Weisheit, die Sie sich gerne in Kreuzstich über die Kodiererkemenate pinnen können, schon vorab:

Objektorientiertes Programmieren versorgt Sie mit einem Baukasten! Sehen Sie vorhandene Klassen immer auch als Möglichkeit zur Erzeugung eigener, spezifischerer Lösungen!

 Ein so gut ausgestattetes Entwicklungssystem wie Xojo verleitet zu der Annahme, stets zur Standardausstattung zu greifen und, sollten Features fehlen, womöglich sehr komplizierte Lösungen anzustreben, zu Drittanbieterlösungen zu greifen oder die Ingenieure zu verfluchen. Mit der Ausnahme der Drittlösungen, die oftmals sehr ausgefeilt sind und einem viele, viele Stunden des eigenen Programmierens sparen können: Tun Sie's nicht! Egal, mit wie vielen Klassen und Controls Sie irgendwann von Xojo ausgestattet werden: In den allerallermeisten Fällen können Sie diese problemlos zu eigenen Speziallösungen verwursten und sich so im Nullkommanüsche einen eigenen Werkzeugkoffer zusammenstellen.

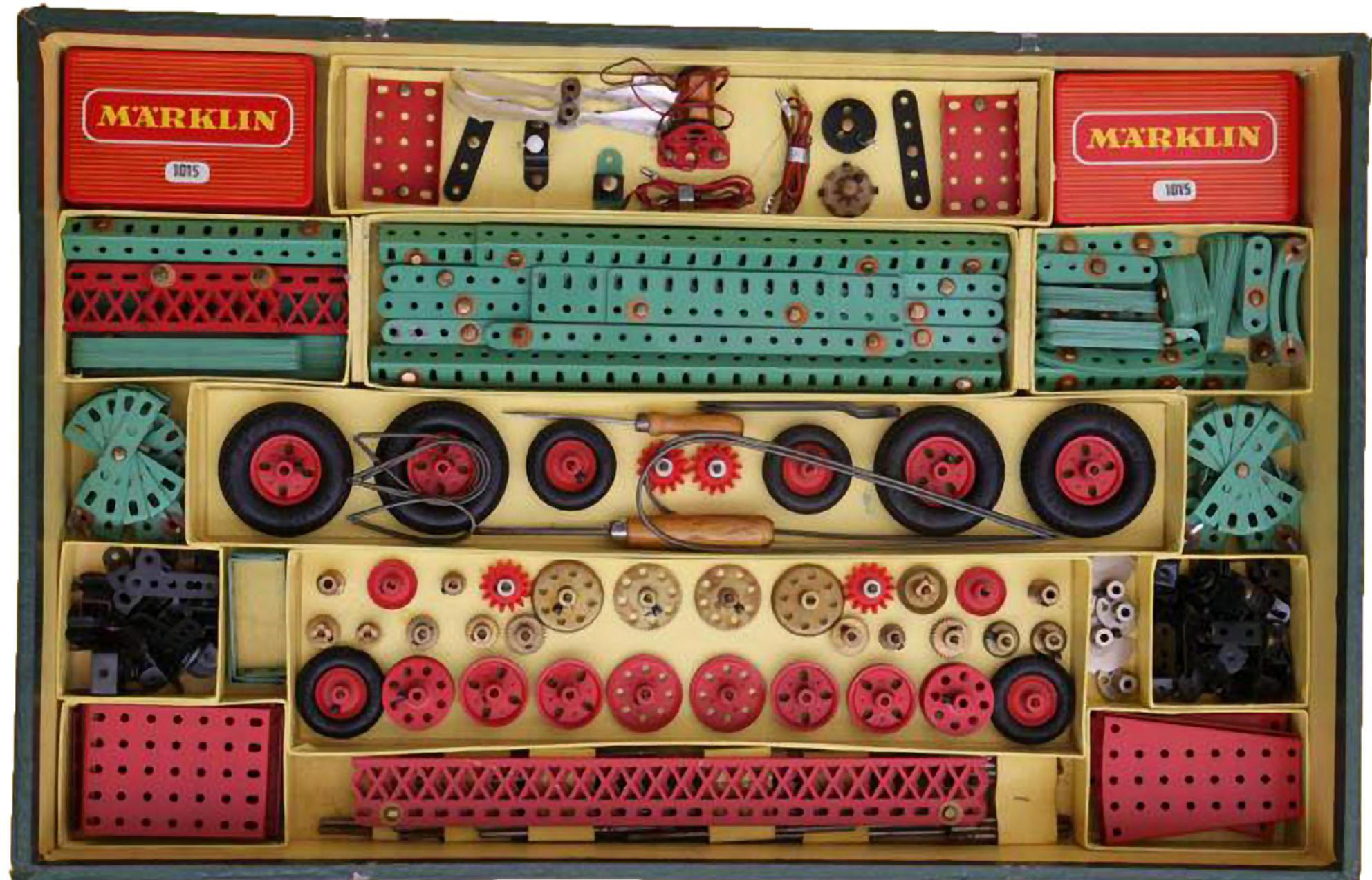


Abb. 11: Sehen Sie's als eine Einladung an den Spieltrieb: OOP heißt: Basteln erlaubt!

@ Gary Higgins, flickr.com



Wir können uns mit den Werkzeugen des objektorientierten Programmierens aber eben nicht nur dröge Strukturen zusammenbasteln, sondern sehr intelligente Küchenmaschinen – fast wie lebende Organismen, die selbstständig auf Reize reagieren können.

Eine Zeitnahmefunktion haben wir im letzten Kapitel zusammengeschraubt. Aber die ist bei strengerer Betrachtung ein ziemlich grobes Werkzeug. Wir müssten allüberall, wo wir wieder die Zeit nehmen wollen, ein paar Zeilen Code reinquetschen – immer wieder die Startzeit nehmen, irgendwas tun, die Endzeit bestimmen, die Differenz berechnen – Ach nö! Redundanzen im Code vermeiden, das sei unser Begehr!

Bislang haben wir zur Strukturierung Methoden kennengelernt. Damit fiele mir nur ein, die Differenzbestimmung auszulagern – eine Funktion zu schreiben, die Start- und Endzeitpunkt entgegennimmt und die Differenz in Nanosekunden auswirft. Probieren wir uns mal daran ...

Nee, wissen Sie was? Eine ganze Menge wissen Sie schon, und ich glaube, ich sollte Ihnen jetzt nicht mehr alles vorkauen. Versuchen Sie sich mal selbst daran, und wenn Sie mit meiner Lösung vergleichen wollen, dann klicken Sie auf den Schlüssel!

Die Aufgabe wäre also, vollständig formuliert:

- ▶ Fügen Sie zum App-Objekt Ihres Projekts eine neue Methode Nanosekunden hinzu, die Start- und Enddatum als `Xojo.Core.Date.Now`-Objekte entgegennimmt und die Nanosekunden der Differenz zwischen beiden als `Integer` ausgibt.

Sie wissen ja: Viele Wege führen zum erfolgreichen Code. Meine Lösung ist deshalb nur eine von vielen – Sie bekommen keine Punkte für die Verwendung gleicher Variablennamen ;)

```
Function Nanosekunden -  
    (StartDatum As Xojo.core.Date, -  
     EndDatum As xojo.Core.Date) As Integer  
    Dim Differenz As xojo.Core.DateInterval = -  
        EndDatum - StartDatum  
    Return Differenz.NanoSeconds  
End Function
```

- ▶ Nun verändern Sie den Run-Eventhandler bitte so, dass die Funktion Nanosekunden aufgerufen und ihr Ergebnis in der Kommandozeile ausgegeben wird.

Das sieht dann z. B. so aus:

```
Dim StartZeit As Xojo.Core.Date = -  
    Xojo.Core.Date.Now  
Dim TestText As Text = "Ich will einmal -  
    wissen, wie lange Xojo benötigt, um eine -  
    Text-Variablen für diesen Text zu erzeugen!"  
Dim EndZeit As Xojo.Core.Date = -  
    Xojo.Core.Date.Now  
Print Nanosekunden -  
    (StartZeit, EndZeit).ToText
```

Ok, ein wenig eleganter als vorher, aber wir haben immer noch `Start-` und `EndZeit`, die wir als Variablen anlegen müssen.

Variablen, die auf Objekte verweisen, werden im übrigen auch gerne unter einem anderen Begriff bezeichnet:

Referenz

Das lasse ich erst einmal mit dem Hinweis so stehen, dass wir hier ja auf ein Objekt verweisen und nicht den Wert eines Datentypen transportieren. Das wird bald noch wichtig und genauer unter die Lupe zu nehmen sein – an dieser Stelle aber nur der Kurzhinweis, damit Sie nicht über den Begriff stolpern.

Man könnte die Referenzen `Start-` und `EndZeit` im Prinzip ähnlich wie eine Methode auslagern – damit würden wir dann Properties erschaffen, in diesem Fall wohl Properties des App-Objekts, weil wir uns mit unserem Code bisher ja darin bewegen.

Aber auch dann müssten wir beide Zeiten im Code festlegen. Sparen würde das nichts, und eleganter wäre es auch nicht.

Jaja, Sie ahnen es schon längst: Meine kleinen Rhetorik-Standards und das laute Überlegen dienen einem Ziel: Sie sollen hier auf die Idee gebracht werden, dass so eine Situation nach einer eigenen Klasse schreit!

Daher mimen Sie jetzt bitte den Überraschten, wenn ich Ihnen verrate: So eine Situation lässt sich prima mit einer Klasse lösen!

Eine Klasse, das ist eben die Vorlage für ein selbstgebasteltes High-Tech-Küchengerät. Ob Sie beim Basteln dabei auf ein anderes Standardgerät zurückgreifen und ihm noch ein paar Nuten anflanschen, oder ob Sie alls von Grund auf neu zusammenschrauben, das ist Ihnen überlassen. Und ich möchte jetzt eine Klasse zur Zeitnehmung entwickeln. Machen Sie mit?



- Fügen Sie eine neue Klasse in Ihr Projekt ein.

Hups? Das hatte ich Ihnen noch gar nicht gezeigt, oder? Ist aber Absicht, denn Sie sollen sich ruhig die Freiheit nehmen, in Xjos Menüs und Funktionen herumzustreunern. Das Insert-Symbol in der Werkzeugeiste und das Insert-Menü in der Menüzeile sind Ihnen ja keine Unbekannten mehr. Schauen Sie dort mal rein, und Sie entdecken den Befehl „Class“ ...

Build Step	▶
Class	
Class Interface	▶
Database	▶
File Type Set	
Folder	
Module	
Event Handler...	⊜ E
Method	⊜ M
Note	⊜ N
Property	⊜ P
Computed Property	⊜ C
Constant	
Delegate	

Ein Rechtsklick auf das App-Objekt und „Insert ...“ helfen Ihnen jetzt nicht, weil die Klasse nicht zum App-Objekt gehört! Klassen sind unabhängige Bauvorlagen, keine Eigenschaften irgendwelcher Instanzen.

(Das hatten Sie sich bestimmt schon zusammengereimt, oder? Xjos Navigator macht die Strukturen meiner Ansicht nach auch gut nachvollziehbar:)

Ihr App-Objekt ist nichts anderes als eine Instanz der Klasse Application – Anwendung!

Ich muss kurz mal einen melancholisch-philosophischen Gesichtsausdruck aufsetzen – moment! –, um auf folgende Gesetzmäßigkeit hinzuweisen:

Wie im Großen, so im Kleinen!

Uffa! [Hermes³](#) im IT-Geplapper?

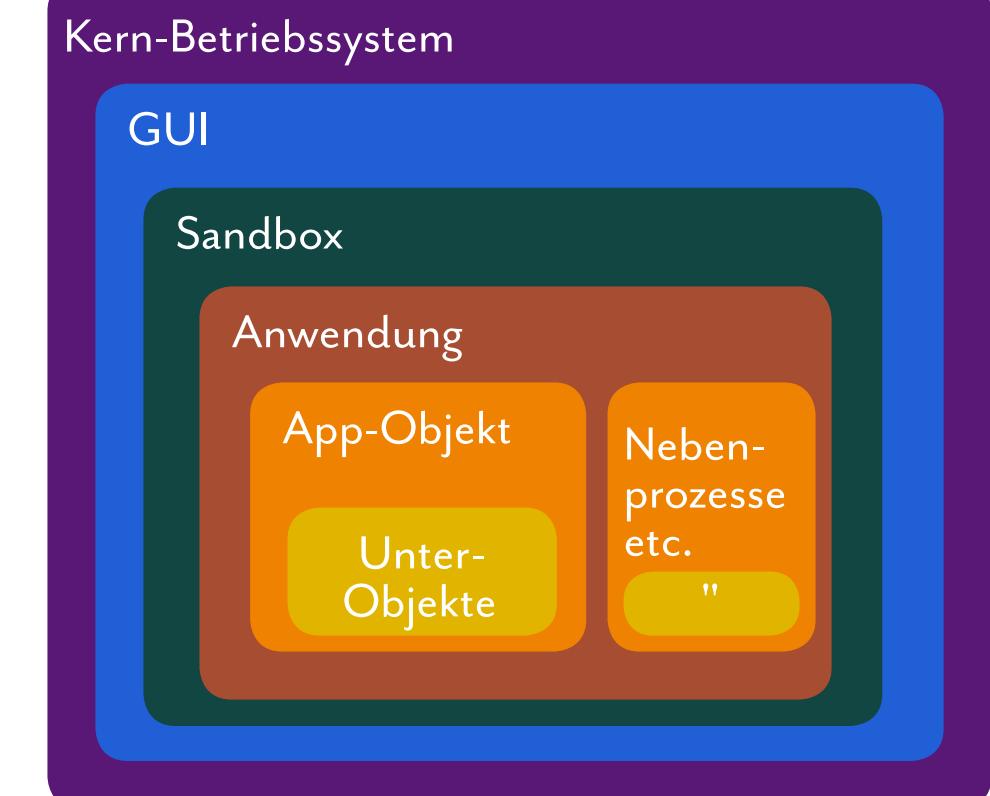
Na wo denn sonst? Der Computer mag bevorzugt binär sprechen, aber er befindet sich im gleichen Universum wie Sie – und unterliegt daher den gleichen Gesetzen.

Ich kann's ja nicht verhehlen – OOP ist keine ganz so triviale Geschichte. Versteht man aber, dass Sie im ganzen Computer Anwendung findet, dann leuchtet auf einmal auch vieles an seinen Eigenarten ein, das vorher nicht so recht einen Sinn formen wollte. Dass ich Sie mit nicht wenigen Fachbegriffen beglücke, ist Ihnen ja schon aufgefallen. Aber viele sind einfach spezifische Begriffe für Sonderfälle der dann doch ewig gleichen Kisten.

Ein Objekt kann jetzt so ziemlich alles sein. Über seine Größe ist damit nichts gesagt. Und da ein Objekt grundsätzlich erstmal immer nur reservierter Speicher ist, dem eine Bedeutung zu-

geordnet wurde, ist es (glaube ich zumindest) ganz sinnvoll, sich mal eines zu visualisieren:

Der Computer, klarer Fall, braucht erst einmal sein Betriebssystem, um einen systematischen Betrieb aufzunehmen. Darin läuft dann ein großer Prozess – auch eine Form von App-Objekt –, der die Verwaltung der Unterprozesse übernimmt. Einer davon ist die graphische Benutzeroberfläche. Viele Anwendungen werden gerne in einen Überprozess gequetscht, eine Sandbox, die überwacht, dass die eigentlichen Apps keinen Unfug mit Sachen treiben wollen, die gar nicht ihr Bier sind. Und zu jedem einzelnen App-Objekt – jeder Anwendung an sich also – können beliebig viele Objekte unter- oder beigeordnet sein. Entsprechend baut sich im RAM eine riesige verzweigte Baumstruktur auf, aber im Grunde sind alles Objekte. Sobald Sie ein paar Klassen entworfen haben, werden Sie auch viel mehr Verständnis fürs Betriebssystem besitzen.





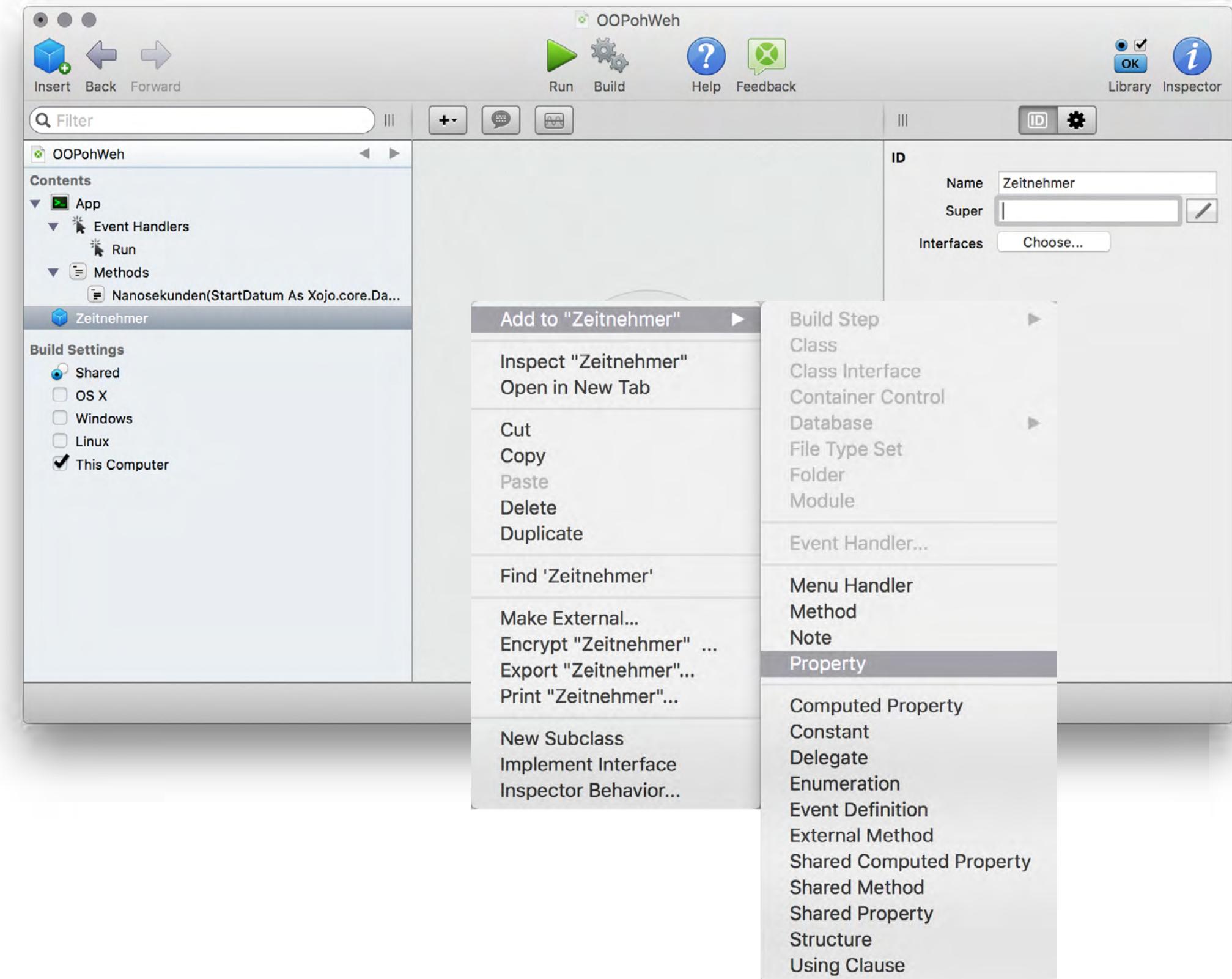
Die Illustration erhebt jeden Anspruch auf sachliche Unrichtigkeit: Selten läuft nur eine einzige Anwendung in der GUI, lange nicht jede läuft überhaupt mit graphischer Unterstützung – sie soll nur veranschaulichen, dass wir eigentlich eine dieser russischen Puppen vor uns haben. Objekt in Objekt in Objekt, aber dazu auch mal neben Objekt unter Objekt ...

Ich bin abgeschwiffen! Zurück zur Klasse! Fügen Sie nun bitte eine neue ein!

Es erscheint ein neues Class-Objekt auf gleicher Ebene wie das App-Objekt im Navigator.

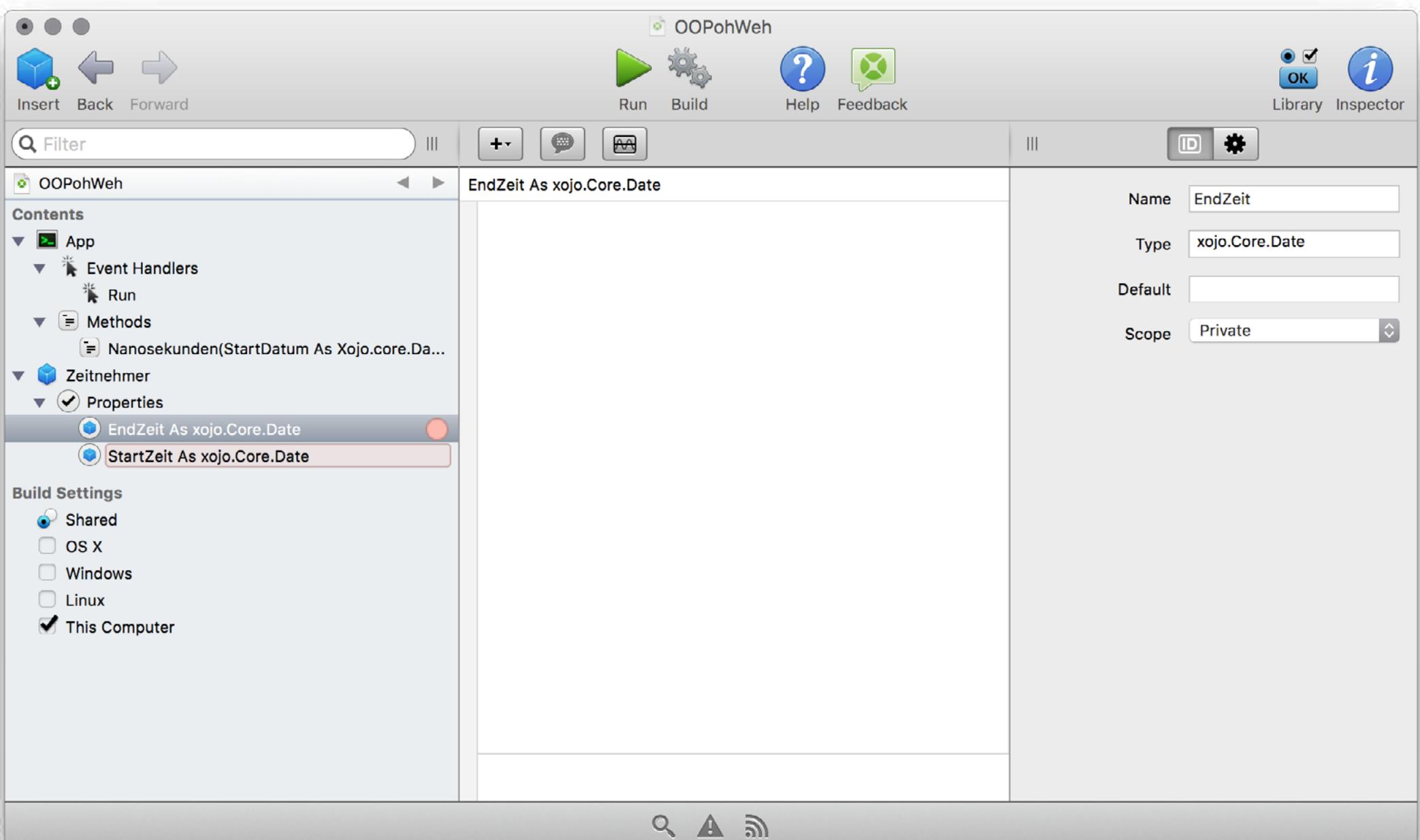
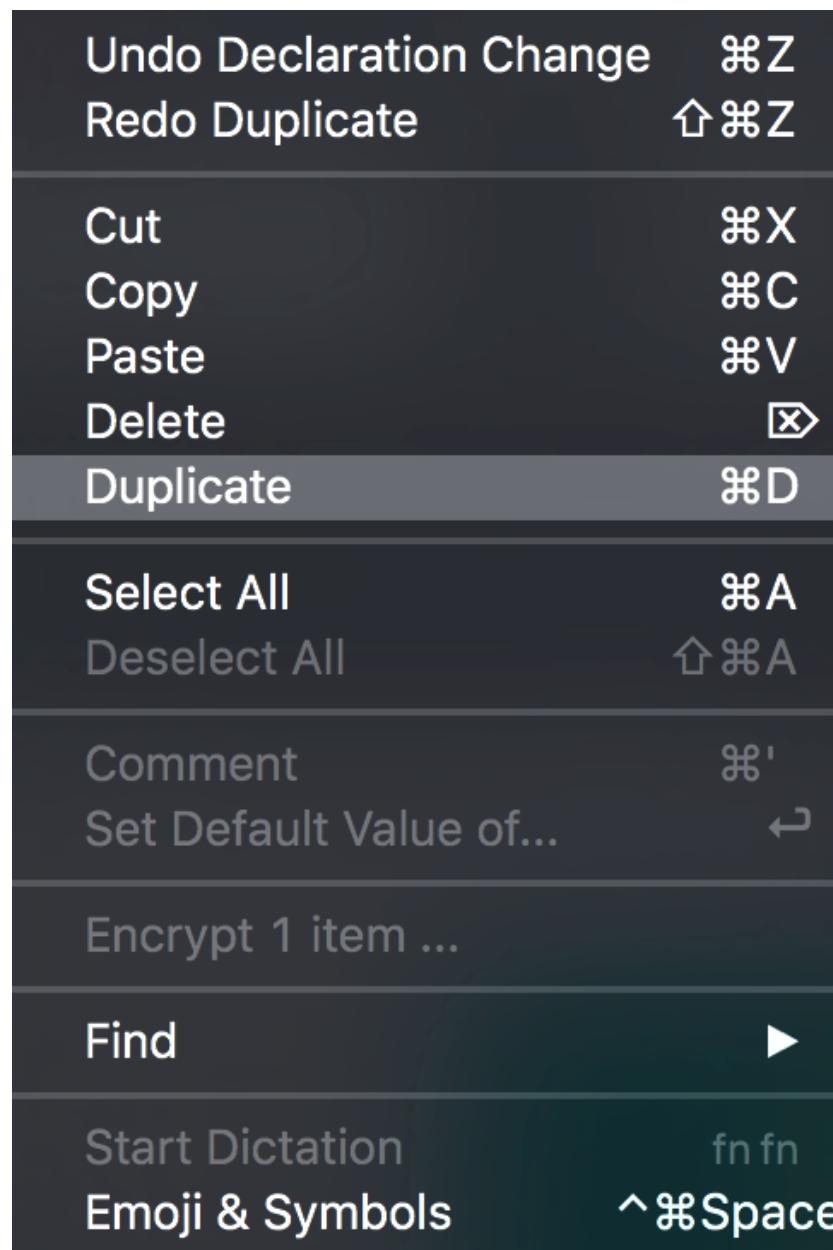
- ▶ Geben Sie ihm doch bitte einen hübschen Namen. Das Feld „Super“ lassen Sie leer. Wir basteln uns eine neue Klasse und bauen nicht direkt auf einer anderen auf.
- ▶ Fügen Sie eine neue Property in die Klasse Zeitnehmer ein. Dazu können Sie jetzt wieder den Kontextklick auf die Zeitnehmer-Klasse im Navigator benutzen oder eine der anderen vielfältigen Insert-Methoden!
- ▶ Benamnen Sie die neue Property im Inspector sinnvoll – z.B. StartZeit, denn Sie soll dazu dienen, die Startzeit des Zeitnahme zu speichern.
- ▶ Geben Sie ihr im Inspector den Typ Xojo.Core.Date.
- ▶ Lassen Sie das Feld „Default“ leer und setzen Sie den Scope der Property im Inspector auf Private.
- ▶ Wiederholen Sie das ganze mit einer privaten Xojo.Core.Date-Property für die EndZeit.

Ihr Projekt sollte jetzt ungefähr so aussehen wie auf der Folgeseite.





Haben Sie die zweite Property ebenso eingegeben wie die erste? Brauchen Sie gar nicht! Sie können Navigator-Elemente **duplicieren**! Den Befehl findet Sie per Rechtsklick oder bei aktiv ausgewähltem Navigator-Objekt in der Menüzeile/Edit. Das Duplikat erhält dann den gleichen Namen, nur mit einem angehängten Zähler, der bei 1 startet.



Warum der **private Scope**? Sie erinnern sich daran, dass ein Prinzip des OOP die Verkapaselung von Objekten ist? Wir wollen nicht, dass irgendein Stück Code von außen den Start- oder Endzeitpunkt verändern kann. Dafür soll nur unsere Klasse verantwortlich sein – das ist dann das Prüfsiegel für korrekte Messungen!

Und noch etwas erinnern Sie vielleicht: den **Constructor**! Bis her haben die beiden Properties den Vorgabewert für ein Objekt – nämlich Nil! Damit kann man schwerlich rechnen, und außerdem soll unsere Klasse ja möglichst komfortabel sein. Daher:



- ▶ Fügen Sie eine neue Methode in die Klasse Zeitnehmer. Sie wissen ja, wie das geht.
- ▶ Nennen Sie die Methode **Constructor** (diesmal müssen Sie den Namen übernehmen!) und lassen Sie das Feld für die Übergabeparameter leer.
- ▶ Achten Sie darauf, dass der Scope der Methode im Inspector auf **Public** gestellt ist.

 Momentan interessieren uns nur zwei **Scope**-Eigenschaften: **Public**, also öffentlich, gibt eine Methode – oder eine Property – für jedermann frei, der Zugriff auf eine Instanz dieser Klasse hat. **Private** beschränkt den Scope auf die Instanz selbst. Später mehr zum Thema!

Geben Sie diesen Code in den Code-Editor des Constructors ein:

```
StartZeit = Xojo.Core.Date.Now
```

Wenn später also eine neue Instanz mittels

```
Dim ZN As New Zeitnehmer
```

deklariert wird, dann trägt sie automatisch ihren Erstellungszeitpunkt ein.

Sparen wir uns schon mal einen Schritt. Dann wär's natürlich praktisch, noch eine Methode zu haben, die die Zeitnahme beendet und uns in einem Abwasch die verstrichenen Nanosekunden liefert. Ergo:

- ▶ Fügen Sie eine weitere öffentliche Methode ohne Übergabeparameter in die Klasse Zeitnehmer. Nennen Sie sie irgendwie in die Richtung **NanosekundenSeitStart** und geben Sie Ihr den Rückgabetyp **Integer**.
- ▶ Füllen Sie den Code der neuen Methode so, dass die End-Zeit festgelegt, das **Xojo.Core.DateInterval** zwischen beiden Zeiten berechnet und seine Nanosekunden ausgegeben werden.

Das hier wäre eine Möglichkeit:

```
Function NanosekundenSeitStart() As Integer
    EndZeit = Xojo.Core.Date.Now
    Dim Differenz As Xojo.Core.DateInterval = -
        EndZeit - StartZeit
    Return Differenz.NanoSeconds
End Function
```

- ▶ Und nun zum krönenden Abschluss von Ganze: Modifizieren Sie den Run-Eventhandler bitte so, dass statt der **Date**-Objekte eine **Zeitnehmer**-Instanz geschaffen und nach Initialisierung des **TestTexts** ihr **NanosekundenSeitStart**-Wert in die Console ausgegeben wird.

Kommen Sie zu einem ähnlichen Ergebnis?

```
Dim ZN As New Zeitnehmer
Dim TestText As Text = "Ich will einmal -
    wissen, wie lange Xojo benötigt, um eine -
    Text-Variablen für diesen Text zu erzeugen!"
Print ZN.NanosekundenSeitStart.ToText
```

- ▶ Starten Sie das Projekt.

Wenn Sie jetzt alles richtig gemacht haben, fallen keine kleinen Kühe und Bäume um und es macht auch nicht Bumm! – vielmehr sollte alles so funktionieren wie bisher, nur mit etwas weniger Code.

Gratulation! Es ist eine Klasse! Sie können jetzt neue Instanzen davon allüberall in Ihrem Projekt erzeugen! Und das tolle ist eben: Wenn Ihnen mal eine sinnvolle Erweiterung einfällt, na, dann setzen Sie sie einfach in die Klasse rein, und schon können alle Instanzen davon den neuen Trick.

Zwei Erweiterungsvorschläge, einfach um Sie auf den Geschmack zu bringen:

Wollen wir eine Instanz von **Zeitnehmer** wiederverwenden, um eine neue Messung zu starten, dann brauchen wir eine Reset-Möglichkeit für den Startzeitpunkt. Wir haben uns ja bewusst Einmischung in die Properties von außen verbeten, um die Messwerte garantieren zu können. Also geht kein

```
ZN.StartZeit = Xojo.Core.Date.Now
```

irgendwo. Versuchen Sie es ruhig. Der Compiler wird Ihnen schon Bescheid geben, dass **StartZeit** eine geschützte Property ist und nur von innerhalb der Klasse gesetzt werden kann. Also hängen wir einfach eine neue öffentliche Methode in unsere Klasse:

```
Sub Reset()
    StartZeit = Xojo.Core.Date.Now
End Sub
```



Fällt Ihnen was auf? Den gleichen Code, der StartZeit auf den gegenwärtigen Zeitpunkt setzt, haben wir schon im Constructor. Und wie war das nochmal mit redundantem Code?

- ▶ Bemühen Sie sich um einen etwas hochnäsigen Gesichtsausdruck, recken Sie Nase und Zeigefinger in die Höhe und skandieren Sie mit mir: „Den versuchen wir zu vermeiden, wo immer es geht!“

Und deshalb optimieren wir den Constructor gleich mit und lassen dies seinen ganzen Code sein:

Reset

 Sie tragen fleißig immer gleich Beschreibungen in das **Descriptor**-Feld im Inspector ein, gelle? Der Methode Reset würde eine Beschreibung guttun, die klarmacht, dass mit ihrem Aufruf eine neue Messung gestartet wird.

- ▶ Hängen Sie jetzt die folgenden Zeilen an den bestehenden Code des Run-Events an:

```
ZN.Reset
Dim Test as New xojo.Core.DateInterval(1, 1)
Print zn.NanosekundenSeitStart.Text
```

Wurde die Text-Erzeugungszeit getestet, setzen wir also unsere Zeitnehmer-Instanz zurück und schauen mal, wie sich im Vergleich ein Xojo.Core.DateInterval schlägt.

- ▶ Testen Sie jetzt mal. Ganz schöner Unterschied, oder? Die Gründe dafür – und wie Sie vieles schneller hinbekommen – folgen mal wieder später!

Vorschlag 2:

Vielleicht bekommen wir doch mal mit komplexen Methoden zu tun, die sogar im Sekundenbereich zu buche schlagen. Eine Nanosekunde ist eine Milliardstel Sekunde. Wir könnten doch, rein vorsichtshalber, die Sekunden als Nanosekunden mit dazu aufschlagen. Der Zahlenraum eines Int64 geht weit über die maximalen 60 Milliarden - 1 hinaus. Also ändern wir doch einfach die Methode

```
Function NanosekundenSeitStart() As Int64
    EndZeit = Xojo.Core.Date.Now
    Dim Differenz As Xojo.Core.DateInterval = ¬
        EndZeit - StartZeit
    Dim Total As Int64 = 1000000000 * ¬
        Differenz.Seconds + Differenz.NanoSeconds
    Return Total
End Function
```

Und schwupplich: Dauert's fortan mal länger, müssen Sie sich keine Gedanken machen, dass die Sekunden übersehen werden könnten.

 **Punkt vor Strich** gilt in Xojo ebenso wie sonst auch in der Mathematik. Es werden also in dieser Rechnung immer erst die Sekunden mit einer Milliarde multipliziert und dann die Nanosekunden addiert.

 Wenn Ihnen noch mehr Erweiterungen einfallen: Nur zu! Sie können ja einmal überlegen, ob auch Minuten noch in den Nanosekunden eines Int64 unterzubringen wären. Usw. Und, ganz verwegen: Finden Sie auch einen Weg, die komplette Differenz darzustellen, bis hin zu Jahren? Jeder Integerbereich wird damit überschritten, so viel kann ich verraten ...

In jedem Fall haben Sie an dieser Stelle wieder einmal den Hinweis auf eigene Explorationen verdient! OOP ist kein einfaches Thema, und insbesondere Programmierern mit linearem Programmier-Background fällt das Umdenken mitunter schwer. Mitunter sehr schwer – ich spreche da aus eigener Kopfzerbrech-Erfahrung.

Sollten Sie von Symptomen dieser Art geplagt werden: Irgendwann ist man damit durch, und dann ist das ganze Thema nur noch digitale Peanuts. Versprochen!

Und falls Sie keine negativen Auswirkungen verspüren: Freut mich riesig! Gönnen Sie sich trotzdem etwas Abwechslung! In den Startprojekten und Tutorials finden Sie auch mal spaßigere Funktionen, und auch wenn dort eventuell Unbekanntes auf Sie lauert: Schauen Sie mal, was Sie vom Code der Einführungs-Demos jetzt schon nachvollziehen können.



3.5. **(Mit-)Teilen macht Freu(n)de!**

Sie haben jetzt schon ein paar Möglichkeiten des Objekt-Baukastens kennengelernt. Unsere Zeitnehmer-Klasse besitzt eigene Properties vom Typ Xojo.Core.Date und die Methoden Reset sowie NanoSekundenSeiteStart und dazu noch einen Constructor, der die Reset-Methode aufruft.

Eventuell haben Sie mittlerweile auch noch weiter experimentiert und ein paar eigene Klassen entworfen oder in die Beispiele hineingeschaut. Dann werden Sie gemerkt haben: Mit diesen Möglichkeiten kommen Sie schon ganz schön weit. Zumal Sie z. B. Properties eines eigenen Objekts auf eigenen Klassen basieren lassen können. Auch wenn mir gerade keine sinnvolle Verwendung dafür einfällt: Sie können also einer Klasse MeineNeueKlasse eine Property Zeitmesser (oder welcher Name auch immer Ihnen gefällt) As Zeitnehmer zuweisen. Das funktioniert problemlos.

 **Achtung Achtung!** Das funktioniert problemlos nur dann, wenn Sie **nicht vergessen**, **Objekt-Properties** auch korrekt **zu instanziieren!** Das können Sie z.B. im Constructor von MeineNeueKlasse erledigen, oder wann immer Sie die Property benötigen. Das haben Sie jetzt schon geahnt, gell: Obiges sagt sich leicht, vergisst sich aber ebenso schnell wieder! Wann immer Ihnen der Debugger eine NilObjectException um die Ohren haut: Schauen Sie mal nach, welche Objekt-Property Sie versehentlich nicht instanziert haben – wo also ein New fehlt. Oder, andernfalls, wo Sie vergessen haben, ein Objekt auf Nil zu prüfen und es einfach drauflosverwenden wollen.

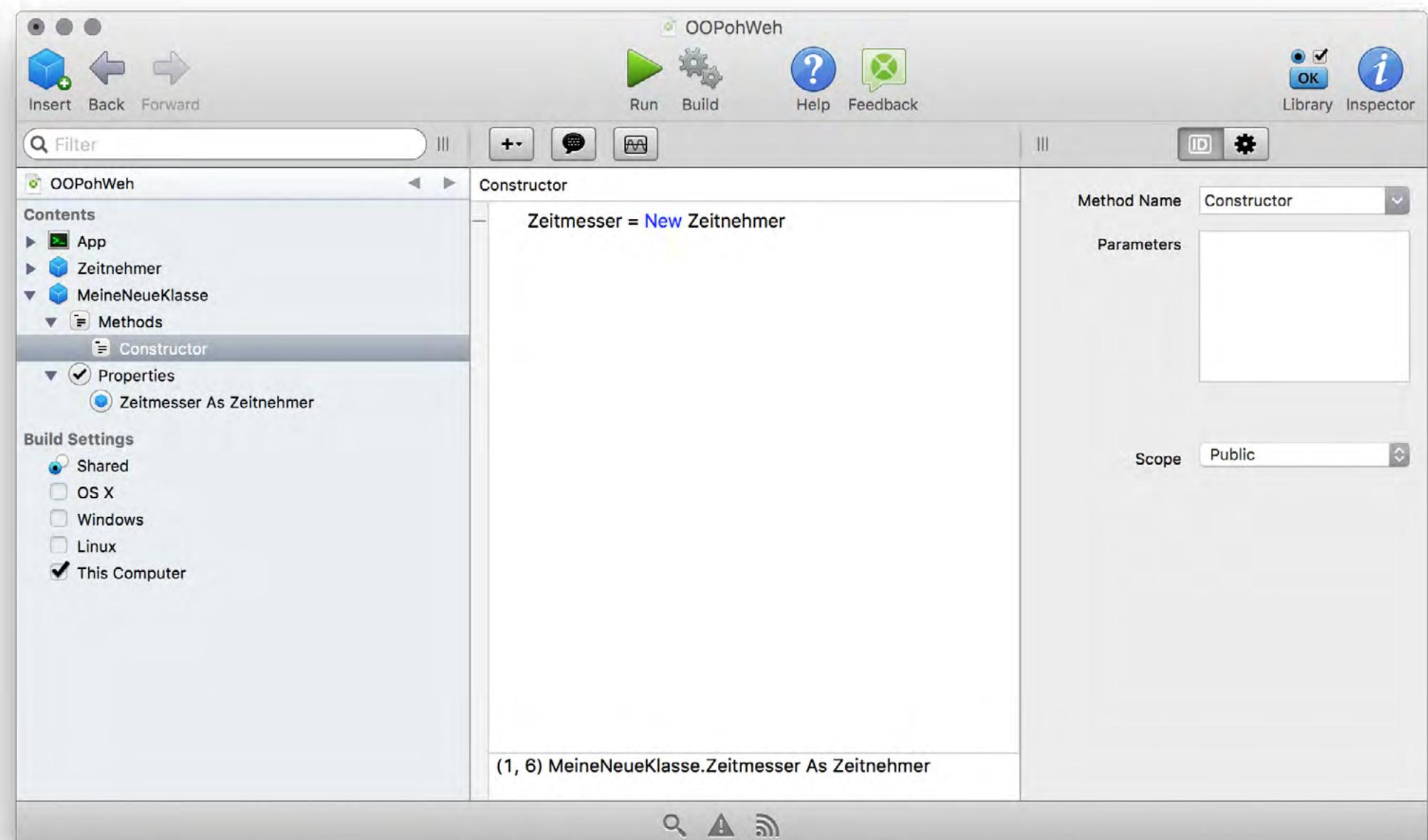


Abb. 12: Nur zur Illustration und als Anregung für ein morgendliches Mantra:
Eine Objekt-Property einer Klasse muss korrekt instanziert werden!



Wie gesagt: Die `MeineNeueKlasse` müssen Sie nicht nachbauen – Sie werden noch oft genug über dieses System stolpern, versprochen!

Bleiben wir noch ein wenig bei unserem `Zeitnehmer`, um mit ihm ein paar weitere Objekt-Features kennenzulernen:

Um einen halbwegen plausiblen Ansatz für eine neue Funktion zu begründen: Nehmen wir mal an, wir wollen wissen, wie oft wir eine `Zeitnehmer`-Instanz eigentlich benutzt haben. – warum auch immer. Das Wissen dazu haben Sie, ergo:

- ▶ Entwerfen Sie eine Property, die zählt, wie oft die `NanoSekundenSeitStart`-Funktion einer `Zeitnehmer`-Instanz aufgerufen wurde.

Das hier wäre eine Möglichkeit:

- ▶ Fügen Sie in die Klasse `Zeitnehmer` eine Property `Aufrufe` As `UInteger` ein
- ▶ Ergänzen Sie die Funktion `NanoSekundenSeitStart` zu dieser Variante:

```
Function NanosekundenSeitStart() As Int64
    EndZeit = Xojo.Core.Date.Now
    Dim Differenz As Xojo.Core.DateInterval = -
        EndZeit - StartZeit
    Dim Total As Int64 = 1000000000 * -
        Differenz.Seconds + Differenz.NanoSeconds
    Aufrufe = Aufrufe + 1
    Return Total
End Function
```

Das war einfach, oder? Welchen Zahlendatentyp Sie für die Property nehmen, ist Ihnen überlassen und der Einschätzung, wie oft diese Funktion denn maximal innerhalb eines Programmlaufs verwendet werden kann. Bei einem Zähler, der niemals negativ werden kann – das Minimum ist ja 0 –, und bei stets glatten 1er-Zählschritten würde ich einen `UInteger` bevorzugen. Falls Sie aber meinen, dass dieser Zähler sehr groß werden kann, dann nehmen Sie ruhig `UInt64` oder gar einen `Double`.

 Wichtig ist natürlich, dass Sie die Zählererhöhung **vor dem Return** vornehmen. Sonst kommt die CPU niemals beim Zählcode an: `Return` heißt ja wortwörtlich „kehre zurück“ – und das macht der Prozess dann auch.

Kleine Denksportaufgabe zwischendurch:

- ▶ Überlegen Sie, ob der Zähler durch seine Konstruktion das korrekte Ergebnis garantieren kann.

Und hier die Auflösung:

Das kann er nicht. `Aufrufe` muss als Property nach „außen“ verfügbar sein – wir müssen den Wert ja auslesen können. Damit könnten wir ihn aber auch von außen verändern. Hier widerspricht die Konstruktion also der VerkapSELungs-Strategie, das das objektorientierte Programmieren verfolgt.

Mit Floskeln der Art „das haben Sie schon geahnt, oder?“ muss ich Ihnen wohl nicht mehr kommen. Deshalb ohne übliche rhetorische Wendungen:

- ▶ Entwerfen Sie eine Lösung, die sicherstellt, dass `Aufrufe` nicht von außen geändert werden kann.

Das mache ich jetzt weniger geheimnisvoll und präsentiere die bisher mögliche Lösung im Klartext: Sie können einfach den Scope von Aufrufe auf Private setzen und eine öffentliche Methode einfügen, die den Wert von Aufrufe returnt.

 Sind Sie drauf gekommen? Hervorragend! Falls nicht, dann setzen Sie das doch kurz mal so um.

Das geht aber noch ein Stückchen eleganter. Wobei auch hier wieder gilt: Ist kein Muss, und eine Methoden-Methode wie oben ist völlig legitim.

Xojo besitzt dieses Feature aber auch eingebaut. Sie sparen ein wenig Entwicklungszeit (und erhalten einen Debug-Vorteil), wenn Sie stattdessen wie folgt vorgehen:

- ▶ Falls Sie die obige Lösung eingefügt haben: Löschen Sie die neue Methode wieder und setzen Sie den Scope von Aufrufe zurück auf Public.
- ▶ Klicken Sie nun mit rechts auf Aufrufe und wählen Sie

Convert to Computed Property

Daraufhin tut sich was im Navigator, und Sie sehen das Ergebnis der nachfolgenden Seite ...



Inspect "Aufrufe"

- Scope ►
- Cut
- Copy
- Paste
- Delete
- Duplicate

Find 'Aufrufe'

Make External...

Encrypt "Aufrufe" ...

Export "Aufrufe"...

Print "Aufrufe"...

Convert to Shared

Convert to Computed Property

▼ Properties

▼ ① Aufrufe As UInteger

- Get
- Set

EndZeit As xojo.Core.Date

① mAufrufe As UInteger

StartZeit As xojo.Core.Date

Aufrufe ist auf einmal als Aufklapp-Property vorhanden, und eine neue private Property mAufrufe ist entstanden.

Wenn Sie Aufrufe durch Klick auf das Dreieck aufklappen, sehen Sie darunter zwei Methoden(!): Get und Set.

- Klicken Sie im Navigator auf die Get-Methode der Computed Property Aufrufe.

Der Code der Get-Methode lautet

```
return mAufrufe
```

Das ist genau das, was die manuell eingefügte Methode vorher gemacht hat, nicht wahr?

Allerdings: Die Property Aufrufe ist immer noch von außen veränderbar, denn ihre Set-Methode lautet

```
Set (value As UInteger)
mAufrufe = value
End Set
```

Das soll ja nicht sein. Und wissen Sie was? Das muss auch nicht so sein!

- Löschen Sie den Code, der in der Set-Methode steht, sodass Ihr Code-Editor leer ist.
- Klicken Sie im Menü Project auf den Eintrag

Analyze Project

(oder benutzen Sie das Tastaturkürzel)

- Run ⌘R
- Run Paused
- Run Remotely ►
- Pause
- Stop Debugging ⌘.
- Step ►
- Breakpoint ►
- Bookmarks ►
- ✓ Break On Exceptions
- Profile Code

Analyze Project ⌘K

Analyze "Zeitnehmer" ⌘⇧K

Analysis Warnings...

Build Application ⌘B

Deploy Application ⌘⇧B

Go To Location ⌘⌘L

Der Compiler prüft jetzt das Projekt, würde es aber nicht starten, falls alles ok ist – was nicht der Fall ist!



Sie werden eine Fehlermeldung erhalten, die Ihnen mitteilt, dass der Property Aufrufe kein Wert zugewiesen werden kann:

Cannot assign a value to this property

sagt der Debugger dazu, und er vermekt die Zeile

```
Aufrufe = Aufrufe + 1
```

in der Methode NanoSekundenSeitStart als fehlerhaft.

Mit der

Computed Property

– zu deutsch „berechnete Eigenschaft“ – haben wir ein Gerüst aus Weitergabenmethoden für eine private Property errichtet. Nach außen hin sieht Aufrufe genau wie jede andere öffentliche Property aus. In Wirklichkeit maskieren sich unter diesem Namen zwei **Methoden**, von denen wir die zweite – die zum Setzen der eigentlichen Property mAufrufe – deaktiviert haben, indem wir Ihren Standardcode ersetzt haben.

Falls Sie beim Fachsimpeln glänzen wollen: Lassen Sie an Stellen wie dieser einfach mal den Terminus

Syntaktischer Zucker

fallen. Denn [genau das](#) haben wir hier vor uns: Die Verwendung von Standardfeatures, die ein Feature vorgaukeln, das in Wirklichkeit eine andere Standardstruktur verfolgt. Alter Wein in neuen Schläuchen also, und eine doofe wörtliche Übersetzung noch dazu, wenn Sie mich fragen. Im Deutschen verstehen wir eine Speise eher mal mit Würze, zumindest im umgangssprach-

lichen, während der Amerikaner gerne floskelhalber Zucker darüber streut. Hier ist also etwas, das vorgaukelt, eine Property zu sein, während in Wirklichkeit zwei Methoden diese Arbeit übernehmen. In der Handhabung ändert sich für den Anwender nichts, außer dass wir nun eine feine Möglichkeit haben, Zugangswege zu beschränken.

☞ Genauso, wie wir den Setter-Code streichen konnten, könnten wir theoretisch stattdessen den Getter-Code entfernen und damit eine Property entwerfen, die nur geschrieben, nicht aber gelesen werden kann. Oder man könnte den Setter mit einer **Plausibilitätsprüfung** des neuen Werts ergänzen. So etwas werden wir uns mit Sicherheit noch anschauen, aber ich wollte es Ihnen für eigene Experimente schon mal mit auf den Weg geben.

☞ Und noch etwas mit auf den Weg: Eine **Computed Property** muss nicht zwangsläufig auf eine private Property verweisen. Sie kann ihrem Namen auch alle Ehre machen und aus anderen Werten einen neuen **Wert berechnen**. Weshalb Sie im Navigator nicht nur eine normale in eine Computed Property umwandeln, sondern auch einfach eine letztere einfügen können – über einen der schon sattsam bekannten Einfügewege. Auch das kommt mit Sicherheit bald, aber ... wissenschon!

Jetzt sollten wir unsere Klasse aber auch funktionsfähig machen. Aufrufe können wir nicht mehr verändern, wohl aber mAufrufe.

► Ändern Sie die fehlerhaft markierte Zeile in der Methode NanoSekundenSeitStart daher in

```
mAufrufe = mAufrufe + 1
```

Wenn Sie das Projekt jetzt mal analysieren lassen, sollte der Compiler wieder grünes Licht geben.

- Starten Sie es aber noch nicht, sondern setzen Sie einen Breakpoint an den Beginn der Return Total-Zeile. Falls Sie vergessen haben, wie das geht: Klicken Sie auf die graue Linie am Anfang der Zeile im Code-Editor.
- Starten Sie das Projekt jetzt.

Erwartungsgemäß läuft es und schaltet in den Debugger. Ein genauerer Blick auf der Folgeseite ...



... zeigt den nebenstehenden Debugger-Status. Sie sehen in seinem rechten Panel die Variable

Self.

Das Selbst hier ist unsere Zeitnehmer-Instanz. Das erkennen Sie einerseits daran, dass diese Klasse hinter dem Self als Datentyp erwähnt wird, und Sie sehen Zeitnehmer unter dem

Stack-Panel

auf der linken Seite als oberstes Objekt und unterlegt aufgelistet.

Mit dem Stack hat es folgende Bewandnis: Unser Programm startet ja im Run-Event des App-Objekts. Aus diesem heraus wird dann der Zeitnehmer instanziert und seine NanosekundenSeitStart -Methode aufgerufen.

Diese soll beim Return wieder zurückkehren. Der Rechner muss sich bei jedem Methodenaufruf merken, von wo aus dieser erfolgte, damit er auch zielsicher den Weg zurück findet. Das macht er in Form eines **Stapels**: Er merkt sich die Rücksprungpositionen im Code. Kommt eine dazu – etwa, indem NanosekundenSeitStart selbst eine weitere Methode aufruft –, wird die aktuelle Adresse obendrauf gelegt.

Jedes Return nun sorgt dafür, dass zur obersten Rücksprungadresse des Stapels zurückgesprungen und diese oberste „Karte“ wieder gelöscht wird.

Nach diesem Ausflug:

- Zeitnehmer hinter Self im Variablen-Panel ist ja unterlegt.
Klicke mal drauf.

The screenshot shows a debugger window with the following details:

Code View:

```
Function NanosekundenSeitStart() As Int64
    EndZeit = Xojo.Core.Date.Now
    Dim Differenz As Xojo.Core.DateInterval = EndZeit - StartZeit
    Dim Total As Int64 = 1000000000 * Differenz.Seconds + Differenz.NanoSeconds
    mAufrufe = mAufrufe + 1
    Return Total
End Function
```

Stack View:

Stack	Main Thread
Zeitnehmer.NanosekundenSeitStart	
App.Event.Run	

Variables View:

Name	Value
Globals	xojo.Core.DateInterval [&h200CFB11]
Differenz	xojo.Core.DateInterval [&h200CFB11]
Return	0
self	Zeitnehmer [&h6002FB11]
Total	39062



Sie sollten jetzt in etwas das hier sehen:

Variables		self As Zeitnehmer
Name	Value	
Aufrufe	1	
EndZeit	xojo.Core.Date [&hCFB11]	
mAufrufe	1	
StartZeit	xojo.Core.Date [&h800DFB11]	

Dies ist nun endlich die Auflösung der Ankündigung, die ich Ihnen anfänglich verlautbarte: Auch wenn eine **Computed Property** eine Variable durch die Hintertür darstellt und in Wirklichkeit Methoden angesprochen werden, taucht sie im Debugger als Variable auf und ihre Daten werden in Echtzeit berechnet. Das ist jetzt hier in unserem Beispiel von wenig Relevanz, weil ja auch mAufrufe als wenn auch private Property angezeigt wird. Bei einer wirklich berechneten Property allerdings ein gravierender Vorteil gegenüber der Get-Methode per Hand wie ganz am Anfang, da Methoden nicht im Debugger als Variablen angezeigt werden können.

3.5.1. Alle für einen!

So Sie meine gutgemeinten Ratschläge befolgen, wovon ich selbstverständlich ausgehe, sind Sie beim Streifen durch die IDE bestimmt schon darüber gestolpert, dass sich auch gemeinsame Properties und Methoden einfügen lassen – erstere ebenso als Computed-Zuckerchen. Nicht nur der Vollständigkeit halber sollten wir uns dieser annehmen, und zwar zunächst in Form der

Shared Properties.

Ich habe die Übersetzung ja schon vorweggenommen: Eine Shared Property wird gemeinsam genutzt, und zwar von allen Instanzen der Klasse zugleich.

Bleiben wir noch einmal bei unserem nun doch ziemlich theoretischen Beispiel der Zeitnehmer-Klasse und nehmen wir mal an, dass es irgendeinen plausiblen Grund gäbe, mehrere Instanzen dieser Methode gleichzeitig zu verwenden. Wie etwa: Ein Zeitnehmer misst die Gesamtdauer einer größeren Methode, während mehrere andere Zeitnehmer die Zeitdauer von Untermethoden feststellen. Okay?

Dann wollen wir nicht nur wissen, wie oft jede einzelne Methode aufgerufen wurde, sondern wir wollen elegant und sicher feststellen können, wie oft die Klasse insgesamt benutzt wurde. Dabei wäre ein manuelles Zusammenzählen der Einzelergebnisse nicht die richtige Methode! Nicht nur, weil sie umständlich und fehleranfällig ist – wir könnten beim Addieren eine Instanz übersehen, oder es könnten temporär Zeitnehmer-Instanzen geschaffen worden sein, die am Ende nicht mehr existieren: Vor allem gehört zu den Grundtugenden des Programmierers die Faulheit, und wenn wir Objekte für uns arbeiten lassen können, ja, dann machen wir das hemmungslos!



Meine nativen Moral-Unterfunktionen zwingen mich zu dieser Stellungnahme: Im letzten Satz war die Rede von Objekten. Niemals nicht von anderen Wesen und dieser Klasse zugehörigen Menschen! Dort sollte es immer ein Geben und Nehmen sein. Der Computer ist eben doch kein vollständiges Abbild der Realität.

Zurück zum Thema:

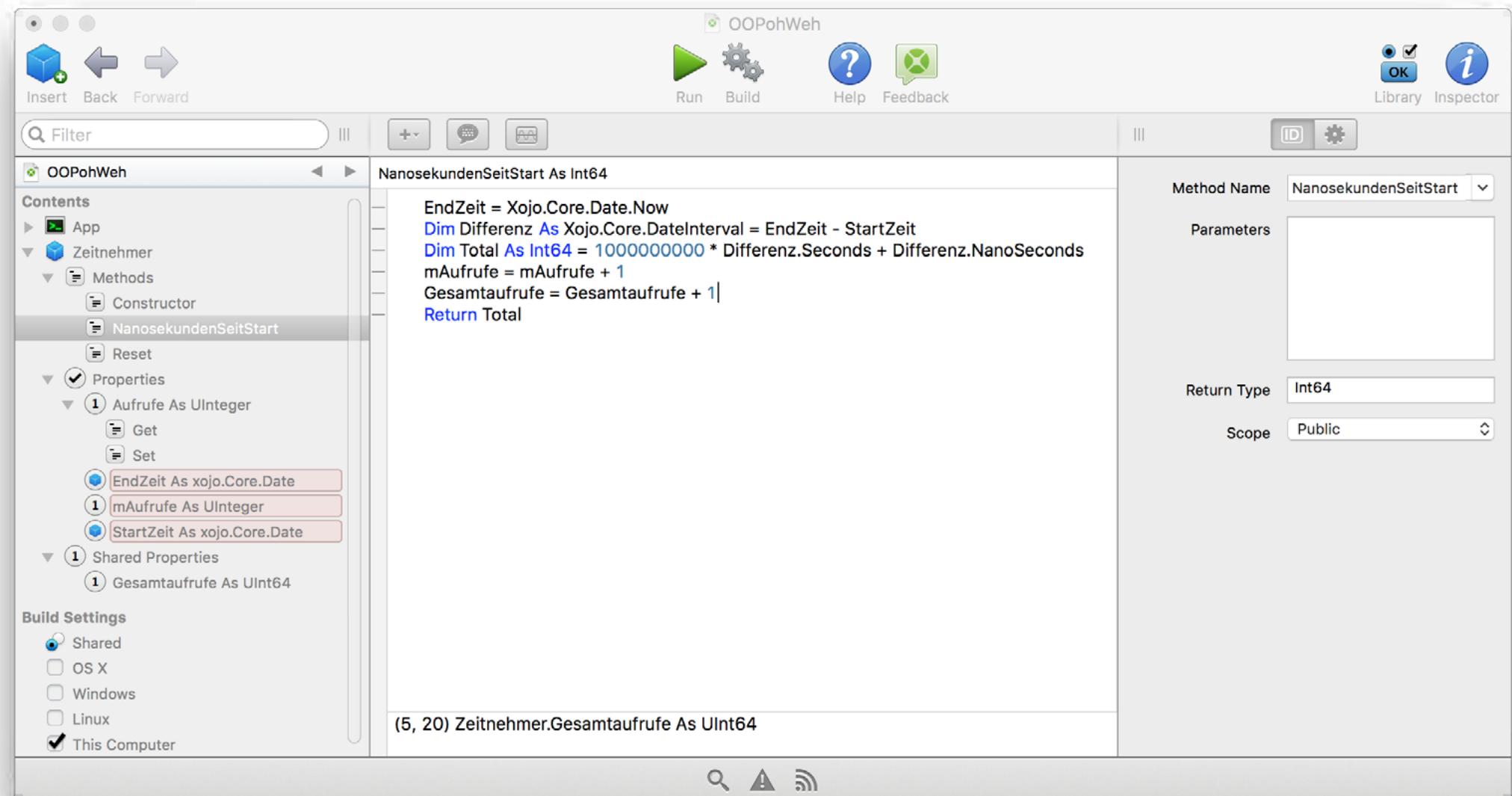
- ▶ Legen Sie eine **Shared Property** an, die als Zähler für die Gesamtanzahl der Funktionsaufrufe von NanoSekundenSeitStart aller Instanzen von Zeitnehmer dient. Einfügen können Sie eine solche Property wie üblich über eine der Insert-Methoden.
- ▶ Erweitern Sie die NanoSekundenSeitStart-Methode, so dass der Klassenzähler bei jedem Aufruf auch um 1 erhöht wird.

Eigentlich können Sie hier vollständig der Anleitung von Seite 71 folgen, nur dass die Property eine gemeinsam genutzte sein muss. Und, da sie die Gesamtzahl aller Aufrufe speichern soll, wäre es vielleicht sinnvoll, sie standardmäßig größer anzulegen als die Instanzzähler – etwa als UInt64 oder als Double. Ich nenne sie mal Gesamtaufrufe As UInt64, und meine Methode NanosekundenSeitStart wird dann noch um die Zeile

```
Gesamtaufrufe = Gesamtaufrufe + 1
```

erweitert – vor dem Return, naturellement.

In voller Schönheit sieht das Programm dann aus wie folgt:



Auch hier hätten wir wieder das Problem, dass GesamtAufrufe eine öffentliche Property und damit von außerhalb manipulierbar wäre. Also

- Konvertieren Sie GesamtAufrufe zu einer **Computed Property**, löschen Sie deren Set-Code und verändern Sie NanosekundenSeitStart, damit die neue private **Shared Property** erhöht wird.

Das Ergebnis, wie nicht anders zu erwarten, ist dann eine

Shared Computed Property,

also eine gemeinsam genutzte berechnete Eigenschaft. Diese greift im Getter wieder auf die private **Shared Property** zurück, und sie kann nur von innerhalb der Klasse verändert werden.

- Den Code für den Run-Eventhandler des App-Objekts diktiere ich Ihnen hiermit:

```
Dim ZN As New Zeitnehmer
Dim TestText As Text = "Ich will einmal wissen, wie lange Xojo benötigt, um eine Text-Variablen für diesen Text zu erzeugen!"
Print zn.NanosekundenSeitStart.ToText

Dim ZN1 As New Zeitnehmer
Dim Aufrufe As UInt64 = ZN.Aufrufe
Print ZN1.NanosekundenSeitStart.ToText
Print ZN.Aufrufe.ToText
ZN1.Reset
Dim Gesamtaufrufe as UInt64 = ZN.Gesamtaufrufe
Print ZN1.NanosekundenSeitStart.ToText
Print Zeitnehmer.Gesamtaufrufe.ToText
```

Nichts aufregendes, oder? Ich habe hier noch einen zweiten Zeitnehmer eingefügt und benutze ihn einfach, um die Zeiten zu messen, die das Abfragen der beiden Properties benötigen.

- Führen Sie das Projekt aus.

Sie werden sehen, dass ganz erwartungsgemäß die Property Aufrufe einen Wert von 1 liefert, während Gestamtaufrufe auch brav alle drei Funktionsaufrufe gezählt hat.

Aber jenseits der Möglichkeit, mit Shared Properties für alle Instanzen zugleich geltende Eigenschaften zu besitzen – und ich hätte Gesamtaufrufe auch so aufrufen können, denn die Klasse selbst wird bei gemeinsam genutzten Objekten zum Ansprechobjekt: –, bieten Shared Properties noch einen weiteren Bonus.

Gesamtaufrufe = Zeitnehmer.Gesamtaufrufe



- Führen Sie das Projekt mehrfach aus und vergleichen Sie die letzten beiden NanosekundenSeitStart-Ergebnisse.

Häufig werden die Werte relativ identisch sein – wie gesagt, man weiß nie so genau, mit welchen Geschichten der Rechner alles so beschäftigt ist, während er offiziell unser Projekt ausführt. Mit einem kleinen bisschen Glück sollten Sie aber bemerken können, dass das Abrufen der **Shared Property** Gesamtaufrufe weniger Zeit benötigt als das Abrufen der **Instanzenproperty** Aufrufe. Falls Sie das nicht sehen können, lassen Sie Aufrufe und Gesamtaufrufe einfach in **Zählschleifen** gleich oft aufrufen und messen Sie die jeweilige Gesamtzeit.

Das wäre also angekündigter Bonus:

Shared Properties und Methoden sind im Vergleich zu Instanzenproperties und -methoden etwas schneller.

Kein Riesenunterschied, aber bei sehr häufig aufgerufenen Features mag sich das summieren.

 Dies ist kein Optimierungsfeature, das sich pur um der Performance willen häufig nutzen lässt. Wenn eine Methode oder eine (Computed) Property auf ein Feature einer Instanz zurückgreift, hätten Sie keinerlei Vorteil davon, dieses kompliziert auf die Klassenebene zu hieven und von dort dann die Instanz zu adressieren. Aber: Wann immer Sie in Ihre Klassen ein neues dieser Objekte einbauen, überlegen Sie sich, ob es auf Instanzen- oder Klassenebene gehört. Und wenn letzteres, dann nur hin damit – Ihr Programm wird, je nachdem, wie oft das Feature benutzt wird, mehr oder weniger stark davon profitieren. Machen Sie sich an dieser Stelle aber noch nicht allzu viele Gedanken darum – wir werden diese Überlegungen in Zukunft ein paarmal gemeinsam hegen.

Der Vollständigkeit halber noch: Habe ich zur Unterscheidung von Instanzen-Objekten gesprochen, dürfen Sie gemeinsam genutzte, also Shared Objekte gerne auch als

Klassenproperties und

Klasenmethoden

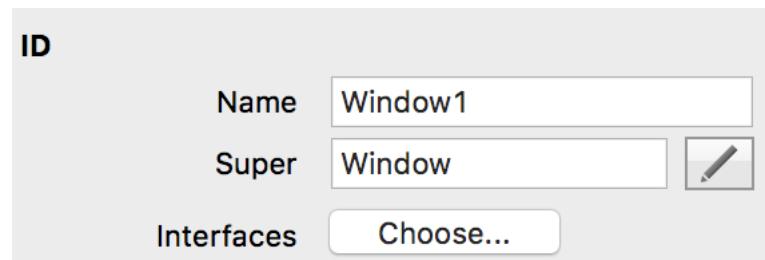
bezeichnen. Das machen andere oftmals nämlich auch so!



3.6. Erben ohne Streit

Eine immens wichtige Eigenschaft von Klassen ist ihre **Vererbbarkeit**. Aus einer Klasse kann eine neue Unterklasse erzeugt werden, die die Features der Mutterklasse erbt – aber auch überschreiben oder modifizieren kann. Klingt kompliziert, ist Ihnen aber wohlvertraut, und zwar von Anfang an:

- ▶ Schauen Sie sich noch einmal das Kapitel „Das erste Programm“ auf Seite 17 an. Wir haben da in einem Desktop-Programm ein **Window** angelegt und auf diesem einen **PushButton** installiert.
- ▶ Vollziehen Sie das Projekt in diesen Anfängen noch einmal nach: Platzieren Sie einen PushButton auf einem Window in einem neuen Desktop-Projekt.
- ▶ Blenden Sie nun in der IDE den Inspector von Window1 ein.
- ▶ Schauen Sie mal:



Window1 besitzt eine Superklasse – Window! Das heißt:

Ein neues Window, das Sie in der IDE anlegen, ist selbst eine neue Klasse.

Der Name der Klasse ist in diesem Fall Window1. Dass Sie davon häufig gar nichts mitbekommen, liegt an der Property **ImplicitInstance** der Window-Klasse (nur im Inspector setzbar).

Diese Klasse wird also implizit instanziert. Weniger hochgestochen: Es wird **automatisch** eine **Instanz** dieser Window-Klasse erzeugt, für die dann projektweit der Name der Window-Unterklasse (also hier Window1) als Variablenname gilt, unter dem Sie die Instanz ansprechen können.

Sie können jederzeit neue Instanzen dieser Klasse erzeugen, wie bei jeder Klasse:

```
Dim W As New Window1
```

Die Klasse Window1 hat alle Features der Window-Klasse geerbt und noch eines dazu erhalten: Das Control **Button1**, das Sie darauf platziert haben.

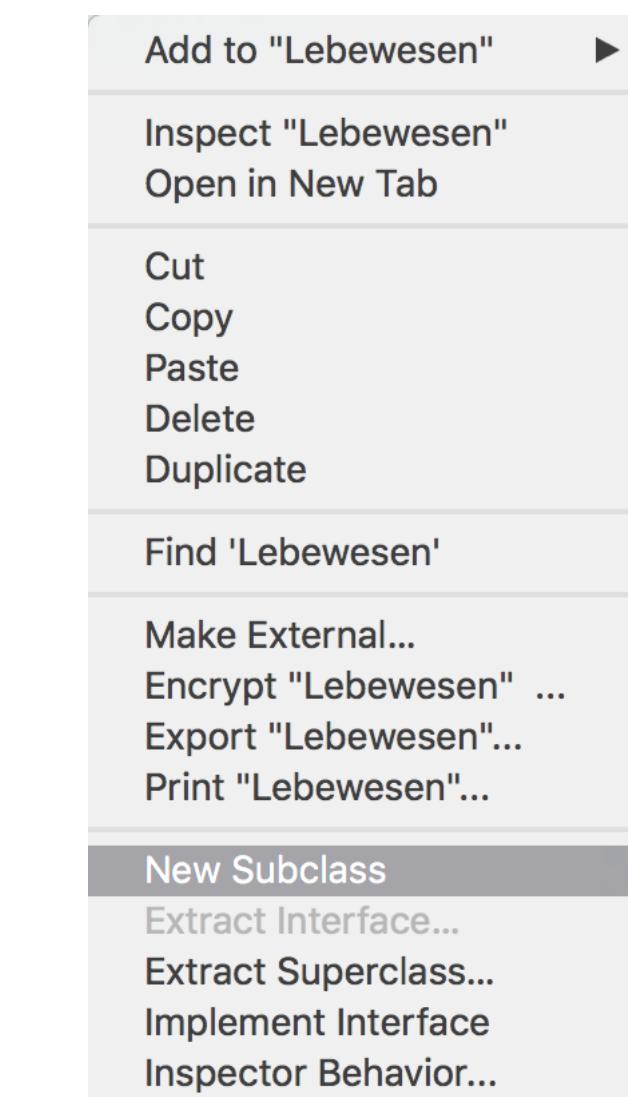
☞ Ein wenig hinkt dieser Vergleich, da es nicht möglich ist, **weitere Unterklassen** von **Window1** als graphisch layoutbare Fenster anzulegen. Das geht nur mit direkten Abkömlingen von Window. Wenn Sie sich vorstellen, dass dies ginge, wäre es einfach zu sehen, dass, so Sie eine Unterklasse von Window1 anlegten, diese dann standardmäßig den Button beinhalten würde, oder?

Schauen wir daher mal in ein besser überschaubares Vererbungsprinzip:

- ▶ Legen Sie ein neues Console-Projekt an und fügen Sie dort eine neue Klasse ein: „Lebewesen“. Geben Sie Lebewesen die Property „Laut“ As Text.



- ▶ Machen Sie einen Rechtsklick auf „Lebewesen“ im Navigator und wählen Sie „New Subclass“:





- Geben Sie der neuen Unterklassie den Namen „Tier“ (standardmäßig nennt Xojo sie erst einmal „CustomLebewesen“)
- Erzeugen Sie auf dieselbe Art **drei Unterklassen von „Tier“**, und zwar „Hund“, „Katze“ und „Maus“.
- Fügen Sie zur Klasse „Hund“ eine **Methode** namens „**Constructor**“ zu, mit dem Code

```
me.Laut = "Wuff"
```

- Was es mit dem **Constructor** auf sich hat, finden Sie unter „Schaffen wir das?“ auf Seite 62 und in der Referenz „Constructor“ auf Seite 140 .
- Machen Sie das gleiche mit den Unterklassen „Katze“ und „Maus“ – z. B. mit den Werten „Miau“ und „Pieps“.
 - Nun geben Sie der Hauptklasse „Lebewesen“ noch eine Funktion „Gebelaut“ mit dem Code

```
Print me.Laut
```

- Wenn Sie aus guten Gründen der Meinung sind, Laute können nur Tiere geben, dann fügen Sie diese Methode auch gerne an die Klasse „Tier“ an. Vergessen Sie dann nur nicht, den folgenden Code entsprechend anzupassen!
- Jetzt nur noch der Code für den Run-Event des App-Objekts:

```
Dim k As New Katze
k.Gebelaut
```

- Führen Sie das Projekt nun aus, und das Ergebnis wird, wenig überraschend „Miau“ sein. Aber wieso?

Die Klasse **Katze** ist eine **Unterklasse von Tier** und **erbt all dessen Features**. **Tier** besitzt davon keine eigenen, hat aber seinerseits als **Unterklasse von Lebewesen** alle Features dieser Klasse geerbt. Dazu gehören die Property „Laut“ und die Methode „Gebelaut“.

K wurde als Katze initialisiert, und der **Constructor** der Klasse Katze sorgt für das richtige Setzen der Property Laut. Die Methode Gebelaut ist allen Lebewesen gemein (sagen wir mal zur Begründung, wir rechnen „Raschel“ und ähnliches mit dazu). Deshalb bringt die Anwendung von Gebelaut das Ergebnis „Miau“ hervor.

Auch wenn Sie sich auf Lebewesen-Ebene unserer Katze K nähern, verändert sich das nicht:

- Ergänzen Sie den **Run**-Eventhandler um die Zeilen

```
Dim l As Lebewesen = K
l.Gebelaut
```

Da K als Katze mittels **New** initialisiert wurde und dabei die Property Laut auf Lebewesen-Ebene gesetzt wurde, bleibt das Ergebnis bei „Miau“.

Dieses Spiel können Sie beliebig weitertreiben. Z. B. stellen sie evtl. fest, dass „Wuff“ nicht für alle Hunderassen die passende Äußerung ist. Dann legen Sie eine **Unterklasse von Hund** namens „Kleinhund“ an mit dem **Constructor**

```
me.Laut = "Wiffel"
```

– oder Sie gehen gleich in einzelne Hunderassen und individualisieren Yorkshire-Terrier, Rehpinscher und wasweißich ... Und können bei der Gelegenheit auch weitere Properties und Me-

thoden ergänzen: durchschnittliche Lebenserwartung, Größe, Anzahl der Beine, ... Überlegen Sie sich dabei, in welche Oberkategorie alles gehört. Machen wir einen Schritt mal gemeinsam:

3.6.1. Auf Identitätssuche

Beine finden wir mit großer Wahrscheinlichkeit wirklich nur bei Tieren. (Jaja, Flimmerhärtchen und Pseudopodien lassen Raum für Diskussion.) Also eigentlich kein Grund, **Beine As Integer** auf Lebewesen-Klasse hinzuzufügen – Pflanzen und Pilze kommen in der Regel gut ohne aus.

- Geben Sie der Klasse **Tier** eine Property **Beine As Integer**.
- Ergänzen Sie die **Constructors** von Hund, Katze, Maus um jeweils

```
me.Beine = 4
```

Und nun stellen Sie sich folgende Situation vor: Igendeine Methode liefert Ihnen eine Variable der Klasse **Lebewesen** zurück. Nun könnte das ein Tier sein und daher Beine haben – zumindest die **Property** Beine besitzt es dann, auch wenn der Wert bei vielen Tieren, die gerne auch mal mit „Sch...“ beginnen, 0 ist – aber es könnte auch eine Pflanze sein, und Beine haben wir nur den Tieren gegeben.

Wir müssen also rausfinden, **ob** unsere Lebewesen-Variable **auch** eine Tier-Variable ist. Umgangssprachlich würden wir auf Englisch einen Satz formulieren, der vermutlich die Worte

„**If my lebewesen is a Tier**“ beinhaltet.



Tja nun – und damit kennen Sie auch schon den Xojo-Befehl für die Bestimmung einer Klasse. Soll die Lebewesen-Variable einfach mal den schönen Namen l tragen, dann wäre der Code

```
If l IsA Tier then
    // Beine berechnen
End if
```

IsA (KlassenName) liefert das Boolesche Ergebnis dafür, ob die Variable der Klasse KlassenName oder einer Unterklasse davon angehört.

Abfragen dieser Art lassen sich also weiter verfeinern:

```
If l IsA Tier then
    If l IsA Hund then
        If l IsA Rehpinscher then
            // Aktion!
        End If
    End If
End if
```

Oder, wenn mehrere Alternativen auf einer Ebene unterscheiden werden sollen:

```
Select Case l
Case IsA Dogge
    // Aktion1
Case IsA Rehpinscher
    // Aktion2
End Select
```

3.6.2. Geworfene Typen

Nehmen wir mal, unser Code hat nun erfolgreich festgestellt, dass l irgendeine Hundeart ist. Nun kommen wir trotzdem nicht an die Beine – dazu brauchen wir ja mindestens ein Tier, und l wurde nun einmal als Lebewesen definiert. Da wir aber sichergestellt haben, dass l mindestens ein Tier ist, können wir es aber auch von der Tier-Seite aus betrachten. Dafür bemühen wir das

Typecasting,

das in etwa folgendes bedeutet: Lege den Mantel der Unterklasse um mein Objekt der Mutterklasse. Hat es sich diesen angezogen, dann können wir auch in die entsprechenden Taschen schauen ...

In Xojo-Code sieht das einfach so aus:

```
Dim T As Tier = Tier(l)
```

Also der Name der Unterklasse, und als Parameter das Objekt der Mutterklasse.

Jetzt können wir auf **T.Beine** zurückgreifen.

Versuchen Sie versehentlich, eine nicht-mögliche Zuweisung vorzunehmen, werden Sie dafür mit einer

IllegalCastException

belohnt. Etwa durch

```
Dim K As New Katze
Dim H As Hund = Hund(K)
```

3.6.3. Das ist ja super!

Stellen Sie sich nun aber einmal vor, Sie legen jede Menge neuer Unterklassen an. Dann müssen sie jedesmal im Constructor die Initialisierungs-Parameter setzen: Für jede Maine Coon, jede Preußischblau, jede Hunderasse immer wieder Beine = 4. Das nervt irgendwann gewaltig. Wenn wir schon Klassenhierarchien haben – lassen die sich nicht auch für den Aufbau nutzen?

Na klar, sie lassen sich. Dafür gibt es in Xojo das Wörtchen

Super.

Super kann nur innerhalb einer Klasse benutzt werden und bedeutet:

Benutze die Methode der Superklasse.

Am praktischen Beispiel: Lassen Sie den **Constructor** der **Hund**-Klasse die identischen Aufgaben für alle Hunde übernehmen:

```
me.Laut = "Wuff"
me.Beine = 4
```

Und verändern Sie die Constructors der Hunderassen zu

Super.Constructor

(Oder lassen Sie die individuellen Constructors weg. Dann wird automatisch der Hund-Constructor aufgerufen)



Bei den abweichenden Hunderassen-Constructors, etwa dem Rehpinscher, brauchen Sie aber einen Super.Constructor, gefolgt von den individuellen Settings:

```
me.Laut = "Wiff"
```

Diese Zeile muss in diesem Fall **nach** dem **Super-Aufruf** folgen. Andernfalls würde der Hund-Constructor den Laut mit dem allgemeinen Wert überschreiben.

3.6.4. ... und noch superer!

So wie es die individualisierten Constructors zeigten, lassen sich alle **Methoden** der Superklasse in Unterklassen **überschreiben**. Sie können dann immer von innerhalb der Subklasse auf die Super-Methode durch

```
Super.MethodName
```

zurückgreifen.

 **Beim Aufruf einer überschriebenen Methode** wird immer die Methode der **untersten gültigen Hierarchieebene** verwendet.

Um das zu verdeutlichen:

► Geben Sie der Klasse Katze eine eigene Methode "Gebelaut"() As Text mit dem Code

```
Print "Schnurr"
```

► Lassen Sie den Run-Eventhandler der App wie folgt lauten:

```
Dim k as new Katze  
k.Gebelaut  
Dim l as Lebewesen = K  
l.Gebelaut
```

► Führen Sie das Projekt aus.

Falls Sie erwartet haben, einmal "Schnurr" und einmal "Miau" zu sehen: Lesen Sie die Klammer-Anmerkung in der linken Spalte unten noch einmal.

Sie können in diesem Fall nicht „nach oben“ typecasten:

Gleichartige Methoden von Unterklassen überschreiben ihre Super-Methoden.

Mit **Super** können die Unterklassen-Methoden aber auf die überschriebenen Methoden zurückgreifen. Wenn Sie den Katzen-Gebelaut-Methodencode mit

```
Super.Gebelaut
```

überschreiben, ist alles wieder ganz normal. Sie könnten im individuellen Unterklassen-Code aber auch die Ausgabe randomisieren – mal Super aufrufen, mal den eigenen Code –, oder was auch immer.

 **Wohlgemerkt:** Das gilt **nur für Methoden**, nicht für Properties!

Wie es um letztere bestellt ist, schauen wir uns auf den Folgeseiten an. Zuvor wieder einmal eine Aufforderung zur Nachwirkzeit:

 War das Neuland für Sie oder noch nicht sehr gefestigstes Gebiet, dann spielen Sie an dieser Stelle bitte erst einmal mit eigenen praktischen Klassenhierarchien weiter. Schauen Sie sich Beispielprojekte an oder überlegen Sie, was es in Ihrer Heimstatt zu kategorisieren gäbe. Entwerfen Sie Klassen dafür und entwickeln Sie ein Gefühl für die Vererbung und den Super-Gebrauch.

 Sollten die Dinge anfangen, schwer durchschaubau zu werden: **Ein skizziertes Schema der Vererbungen hilft ungemein!**

An sich ist es aber gar nicht kompliziert, oder? Es wird auch nicht allzu viel schweriger, aber für die Folgeseiten sollten Sie sich bis hier relativ sattelfest fühlen. Deshalb folgt auch einmal kurz die Fortgeschrittenenfarbe. Falls Ihnen das zu viel Grundlagenwissen auf einmal ist: Springen Sie gerne zu einem anderen Kapitel und kommen Sie wieder, wenn Sie merken, dass Ihr Wissen über Klassen noch Fragen offenlässt.



3.6.5. Eigentümlichkeiten von Eigenschaften

Schauen wir uns nun wie angekündigt mal die Properties an. Was passiert, wenn eine Subklasse eine Eigenschaft erhält, die ihre Superklasse bereits besitzt?

- ▶ Geben Sie der Klasse **Hund** eine eigene Property Laut As Text.
- ▶ Ändern Sie den Run-Eventhandler zu

```
Dim h As New Hund  
h.Gebelaut
```

- ▶ Führen Sie das Projekt aus und ...
... Schweigen im Walde!

Wenn Sie sich die Compilermeldungen anschauen, finden Sie einen Warnhinweis, der den Grund erklärt:

This property shadows one already defined by (Superklasse)

und als verdächtig markiert wird dabei die Property Laut in der Klasse Hund.

Das ist angemerkerter Unterschied beim Vererbungsverhalten zwischen Properties und Methoden:

Eine Property einer Unterklasse überschattet die gleichartige und gleichnamige Property der Superklasse.

Der **Constructor** von Hund hat mit `me.Laut = "Wuff"` also die Property Laut gesetzt, die die **Klasse Hund** separat für sich besitzt. Die Methode **Gebelaut** existiert aber auf Lebewesen-Ebene und benutzt die Property Laut **der Superklasse**.

Diese wurde vom Constructor nicht gesetzt, deshalb ist Hund h sprachlos. Der große Unterschied ist:

Eine überschattete Property existiert zweimal: Einmal auf Superklassen- und einmal auf Subklassen-Ebene.

Der Code `me.Laut` im Constructor von **Hund** bezieht sich jetzt auf die Property Laut, die in der **Klasse Hund** angelegt ist.

`Super.Laut` funktioniert nicht – Super bezieht sich nur auf Methoden. Wir kommen aber trotzdem an die Ursprungsproperty.

- ▶ Ändern Sie den Code im **Constructor** der Klasse **Hund** auf

```
Lebewesen(me).Laut = "Wuff"
```

Typecasting funktioniert also in beide Richtungen: Von der Superklasse zur (gültigen) Unterklasse und von einer Unterklasse zu einer beliebigen Oberklasse.

Wenn Sie das Projekt ausführen, sehen Sie, dass der Hund seine Stimme zurückerhalten hat.

3.6.6. Zusammenfassender Ausblick

Designtechnisch ermöglichen die Vererbungsstrategien des objektorientierten Programmierens also einiges, werfen aber auch Fragen und Probleme auf. Es hat seinen guten Grund, dass der Compiler einen Warnhinweis ob der überschatteten Property ausspuckt: Es bedarf größerer Planungsvorbereitung und sehr guter Dokumentation, um in diesem Fall den Überblick zu behalten. Welcher Laut soll's denn sein? Einfache Regel:

Versuchen Sie, überschattete Properties zu vermeiden.

Es mag Fälle geben, in denen sich ihr Gebrauch nicht ganz vermeiden lässt. So ganz ohne Grund wurden sie schließlich nicht eingeführt. Wenn überschattete Eigenschaften aber täglich in Ihrem Code vorkommen, und erst recht, wenn Sie sich regelmäßig informieren müssen, auf welcher Ebene Sie denn nun gerade auf sie zugreifen sollen, dann sollten Sie sich Gedanken machen, ob eine andere Lösungsstrategie nicht vielleicht gewinnbringender wäre.

Mitunter ist es manchmal beim Setzen einer Property in einer Unterklasse einfach noch nötig, irgendetwas anderes zu machen, was in der Superklasse nicht nötig war. Dann würde es viel mehr Übersicht bringen, eine **Computed Property** Laut zu benutzen, die auf Lebewesen-Level die Protected Property `mLaut As Text` liest und schreibt.

Unterklassen, die beim Zuweisen oder lesen noch irgendwts anderes erledigen müssen, können das dann einfach in den Gettern und Settern ihrer eigenen Computed Property Laut – zusätzlich zum Zugriff auf die nur einmal „wirklich“ vorhandene Property `mLaut`. Siehe nächste Seite!



Angenehmer Nebeneffekt:

Computed Properties führen nicht zu Überschattungs-Warnhinweisen!

Ein anderes Problem des bisherigen Vorgehens hab ich noch gar nicht angesprochen: Das Setzen der Properties im Constructor. Bei einer überschaubaren Anzahl von Klassen und Eigenschaften hält sich der Aufwand noch in Grenzen. Aber malen Sie sich mal aus, dass wir aus den bisherigen Klassen eine Vielzahl von Unterklassen erzeugt haben – alles mögliche Getier in Einzelbeschreibung, womöglich noch in weiteren Unterklassen sortiert. Und nun imaginieren Sie bitte noch weiter, dass wir dann feststellen, dass wir ganz oben, auf der Tier-Ebene z.B., noch eine weitere Property benötigen, die dann auch wieder individuell in Unterklassen-Constructors definiert wird.

Und nun behalten Sie mal bitte schön den Überblick, welchen Constructor Sie schon bearbeitet haben!

Sie hören's am Anschwellen der Hintergrundmusik: Ich bereite hier wieder möglichst dramatisch ein danndochgarnichtsowil-des Lösungskapitel für ein neues Programmierproblem vor. Und vermutlich werde ich Ihnen wieder erzählen, dass Sie die Lösung eigentlich schon kennen. Wetten?

The screenshot shows a software interface for managing class hierarchies and properties. On the left, a tree view displays the following structure:

- Tier
- App
 - Event Handlers
 - Run
- Lebewesen
 - Methods
 - GebLaut
 - Properties
 - Laut
 - mLaut** (highlighted)
- Hund
 - Methods
 - Properties
 - Laut
 - Get
 - Set**

On the right, a code editor window titled "Set(value As Text)" contains the following C# code:

```
mLaut = value
// mach noch irgendwas individuelles
```



3.7. Geschwätzige Ereignisse

Zumindest mal ein Kontrast zum üblichen Geschwätz über Ereignisse – und erst recht zum Geplapper über Nichtereignisse.

Natürlich ist die Rede von Events. Wie Sie einen **Eventhandler** hinzufügen, das ist kalter Kaffee für Sie. Um aber den eigentlichen Zweck noch einmal zu rekapitulieren:

Ein Eventhandler füllt den Platz einer vorbereiteten Methode, die beim Eintreten eines bestimmten Ereignisses aufgerufen wird.

Wer aber bestimmt das Eintreten eines solchen Ereignisses? Manchmal ist hierfür der Verantwortliche als „das System“ benannt worden. Das klingt so'n büschen nach unfundierter Systemkritik, und deshalb, im Licht der Kenntnis von Objektstrukturen:

Auslöser eines Events ist ein Objekt.

Nein! Doch! Oh! So wie beim Kommandozeilen-Projekt das **App**-Objekt über einen **Run**-Event gebietet. Und da ein Objekt nur eine Instanz einer Klasse ist, muss es auf Klassenebene Möglichkeiten geben, einen Event zu definieren und ihn auszulösen. Und, na klar, die gibt es, und die machen wir uns jetzt zunutze.

Sie erinnern sich an die Tier-Beine? Für die sorgen wir jetzt:

- ▶ Geben Sie der Klasse Tier eine Property Beine As Integer.
- ▶ Wählen Sie die Property im Navigator aus, rechtsklicken Sie und wählen Sie „Convert to Computed Property“.

► Setzen Sie den Default-Wert der Property mBeine auf -1. Das ist natürlich ein unmöglicher Wert für Beine, und der hilft uns, zu bestimmen, ob dieser Wert gültig gesetzt wurde. Bei -1 können wir getrost davon ausgehen, dass das nicht der Fall ist, daher:

► Ändern Sie den **Getter** von **Beine** auf

```
If mBeine = -1 then
    Dim SubBeine As Variant = RaiseEvent HoleBeine
    If SubBeine = Nil then
        mBeine = 0
    Else
        mBeine = CType(SubBeine, Integer)
    End if
End if
Return mBeine
```

Im Getter wird also zunächst geprüft, ob mBeine noch den Vorgabewert -1 besitzt. Falls ja, wird ein **Variant** erzeugt, dessen Wert sich aus dem Event **HoleBeine** ergibt.

Ein Variant kann alles sein: Integer, Double, Date, jede beliebige Klasse ... Deshalb ist sein Vorgabewert Nil, also nicht initialisiert. Oder, zum Mitschreiben:

Variant ist ein Datentyp, der jeden anderen beliebigen Datentyp aufnehmen kann. Sein Vorgabewert ist Nil = nicht initialisiert.

 **Warum nicht Integer?** Integer liefert als Vorgabewert 0, nicht -1. Wenn ich den (gleich zu erstellenden) eigenen Event HoleBeine aufrufe und dieser direkt einen Integerwert liefert, weiß ich nicht, ob eine 0 das Ergebnis eines Eventhandlers oder ein nichtbeantworteter Event ist. Der Umweg über Variant gibt die Möglichkeit, das zu überprüfen.

Die Prüfung erfolgt dann auch gleich. Wurde kein Wert durch einen Eventhandler geliefert, kann ich mBeine auf einen Vorgabewert setzen, der mir anzeigen, dass hier einmal geprüft wurde – also 0. Damit stelle ich sicher, dass die Event-Abfrage wirklich nur einmal passiert.

Andernfalls – SubBeine, mein Variant-Platzhalter für die per Event zurückgelieferten Beine ist nicht Nil – kann ich per **CType** einen **Typecast** des Variants auf Integer erwzingen und diesen Wert in mBeine speichern.

CType (Variable, Datentyp/Objektklasse) As
NeueKlasse

versucht die Umwandlung einer Variablen in eine andere Klasse oder einen anderen Datentyp.

RaiseEvent, das, wie der Name vermuten lässt, einen selbstdefinierten Event auslöst, brauchen Sie gar nicht, wenn Sie keine normale Methode HoleBeine in Ihrer Klasse besitzen. Dann können Sie den Event wie eine Methode aufrufen. Ich finde, RaiseEvent im Code macht die Sache aber noch etwas klarer.



- Wählen Sie die Klasse Tier im Navigator an, rechtsklicken Sie und wählen Sie „Add To .../Event Definition“.

Die jetzt erscheinende Maske im Inspector gleicht fast völlig der einer Methode, nur steht dort zuoberst **Event Name**, nicht Method Name.

- Geben Sie unter Event Name **HoleBeine** und als Rückgabewert **Variant** ein.

Event Name	HoleBeine
Parameters	
Return Type	Variant

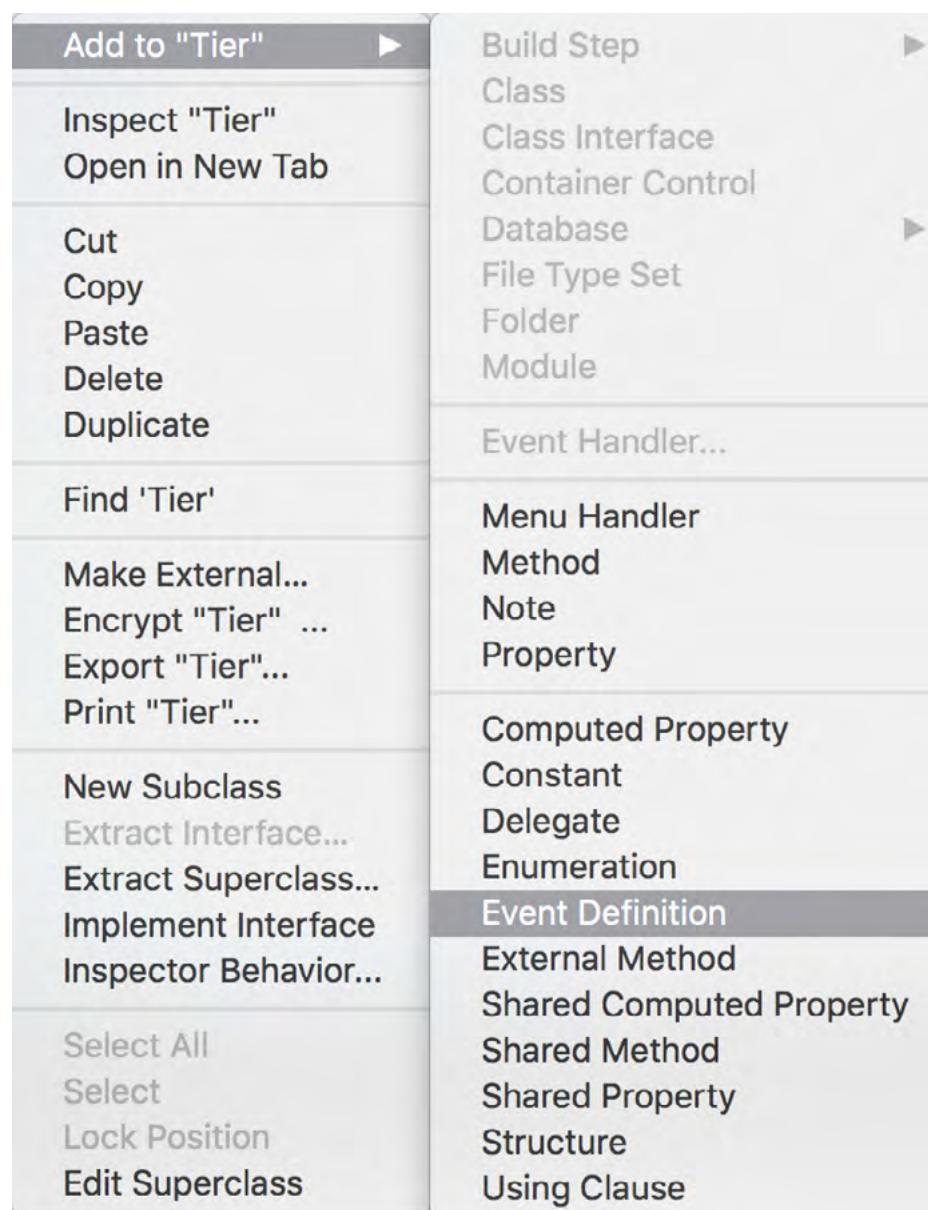
- Wählen Sie nun im Navigator die Klasse **Hund** und fügen dort den **EventHandler** für **HoleBeine** ein:

Return 4

- Lassen Sie den **Run**-Eventhandler des **App**-Objekts mal so lauten:

```
Dim h As New Hund
Print h.Beine.ToString
```

- Nun lassen Sie das Programm laufen und zählen Sie die Beine: 4!



Das mag erst mal ein wenig verwirrend sein, aber es ist gar nicht kompliziert. Gehen wir den Programmablauf mal Schritt für Schritt durch:

- Die Anwendung startet, was den **Run**-Event des **App**-Objekts auslöst.
- Dieser erzeugt eine **Variable h** des Typs **Hund**.
- Der **Getter** der Computed Property **Beine** der Superklasse **Tier**, zu der Hund h gehört, wird aufgerufen.
- **mBeine** ist noch -1, also wird der Event **HoleBeine** ausgelöst.
- HoleBeine findet den passenden **Eventhandler** in der Unterkategorie **Hund**. Dieser liefert den Wert 4.
- HoleBeine erkennt, dass 4 **nicht Nil** ist, und wandelt daher den Variant in einen Integer um und speichert diesen in **mBeine**.
- HoleBeine liefert den Wert von **mBeine** zurück.
- Der Wert 4 wird als Text vom **Run**-Eventhandler aus ausgegeben.

Das ist also der volle Nutzen der Events, die Sie bisher nur als Methode kannten, um mit einem Objekt zu reden:

Mittels Events kann eine Klasse mit ihren Instanzen oder mit ihren Unterklassen kommunizieren.

Dank des Rückgabewerts geht das auch in beide Richtungen. Wobei die Kommunikation immer von der Superklasse aus ausgeht: Nur hier kann der **Event** definiert und mittels RaiseEvent aufgerufen werden. Anders gesagt:

Ein Event ist eine Methode, die eine Klasse in ihrer Instanz oder ihrer Unterklasse aufruft.



Ich hatte Ihnen ja eingangs schon gesagt, dass es gar nicht so viele unterschiedliche Grundprinzipien bei der Programmierung gibt. Hier ist also wieder so ein Fall: Der **Event** ist nur eine **spezialisierte Methode**. Üblicherweise bewegen Sie sich bei Methoden auf Instanzen- oder Klassenebene. Gegebenenfalls können Sie mit **Super**. eine überschriebene Methode der Superklasse aufrufen.

Mit einem Event können Sie sich in die andere Richtung bewegen: Von der Superklasse nach unten.

Selbstverständlich können Subklassen ihrerseits neue Events definieren, und sie können Events auch weiter durchreichen. Nehmen wir mal diesen Fall:

Sie sind sich nicht sicher, ob alle Hunde vier Beine haben. Vielleicht wollen Sie später individuelle Exemplare als eigene Subklassen aufnehmen (oder als Instanzen – der Event ließe sich ja auch per Instanz beantworten!), und da könnten auch dreibeinige Unfallopfer dazugehören. Es wäre also nicht verkehrt, den Vorgabewert flexibel zu halten:

- Rechtsklicken Sie auf den in der Klasse Hund angelegten Eventhandler **HoleBeine** und wählen Sie „**Create Event Definition from Event**“

Damit wird der gehandhabte Event als neue, eigene Eventdefinition in der Klasse Hund angelegt. Sie könnten ihn auch manuell erstellen so wie auf der Vorseite in der Klasse Tier beschrieben. Wenn Xojo aber schon so viele praktische Zeitsparer besitzt: Warum nicht nutzen?

- Ändern Sie den **Eventhandler** HoleBeine der Klasse Hund zu

```
Dim SubBeine As Variant = RaiseEvent HoleBeine
Return If (SubBeine = Nil, 4, SubBeine)
```

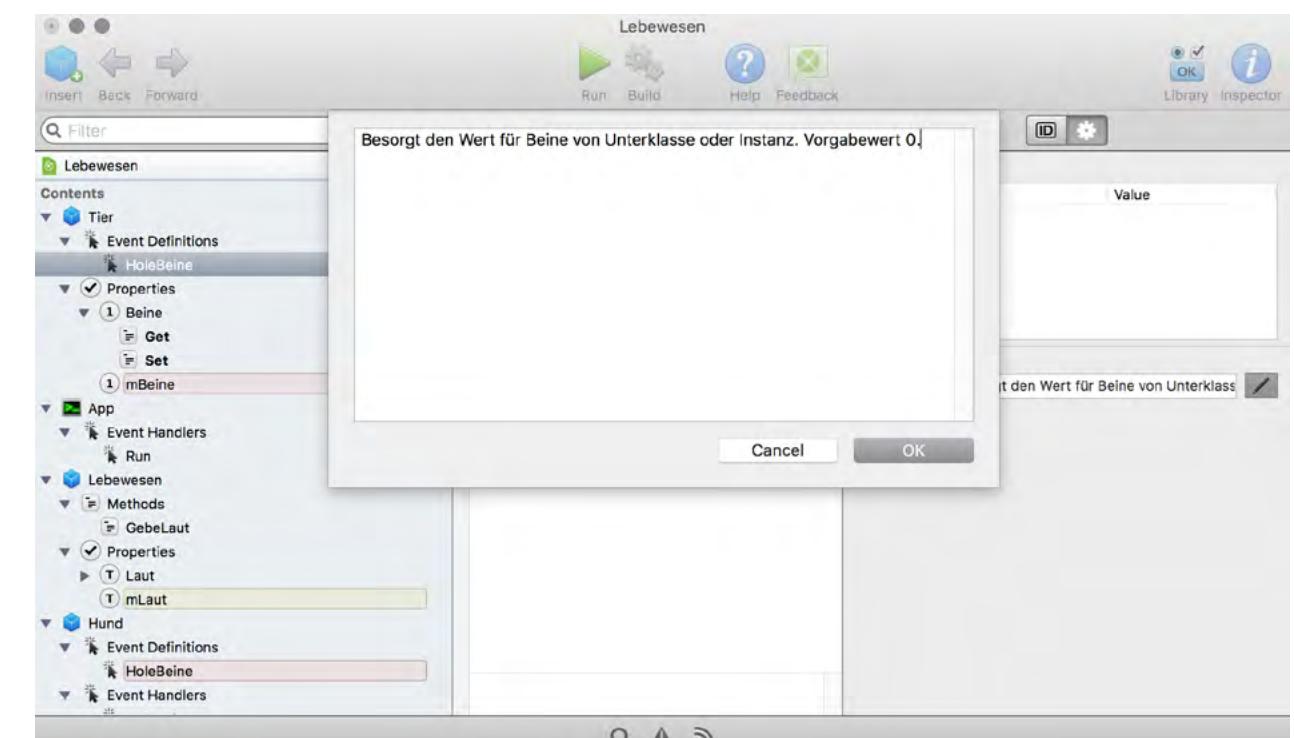
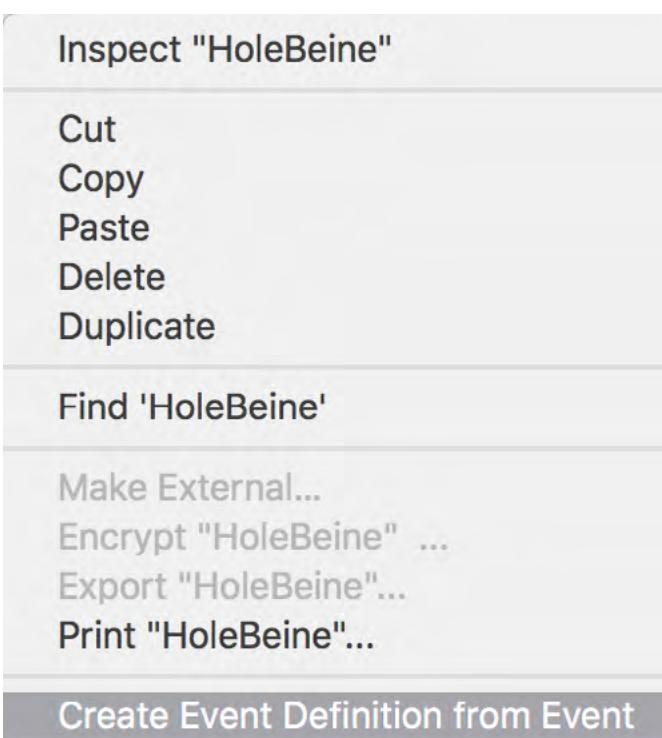
- **Das ternäre If**, falls Sie es vergessen haben, ist die abgekürzte Schreibweise der Bedingung

```
If SubBeine = Nil Then
    Return 4
Else
    Return SubBeine
End If
```

Damit bleibt die Klasse voll flexibel und individualisierbar. Egal, ob Sie später den fünfbeinigen Harrisburg-Terrier als Klasse aufnehmen oder den treuen Hannibal vom Sägewerk als Instanz der Klasse Buntgemischt: Sie haben die Property Beine an einer zentralen Stelle in Ihrer Klassenhierarchie – in mBeine –, und müssen keine großen Verrenkungen unternehmen, was individuelle Abweichungen angeht.

► Wenn Sie schon länger programmieren, aber keine eigenen Eventstrukturen verwendet haben, dann mein Tipp: Suchen Sie sich ein älteres Projekt, in dem Sie komplizierte Kommunikationsstrukturen aufgebaut haben, die mit Events viel einfacher zu lösen wären. Bauen Sie einen Teil auf Eventbearbeitung um, und prüfen Sie, ob Ihr Code davon nicht sehr viel übersichtlicher und einfacher wird.

► In jedem Fall wird Ihr Code von interner Dokumentation profitieren. Eine **Description** (zu erreichen über das Zahnrad-Symbol oben im Inspector) spricht später Bände:





3.8. Klasse sucht Anschluss

Eigene Events bringen viel Komfort im Klassenzusammenhalt, wie soeben beschrieben. Es gibt nur ein Problem: Sie müssen von der Superklasse ausgelöst werden. Nun, im Prinzip kann die Subklasse natürlich eine Methode der Superklasse aufrufen, die ihrerseits einen Event startet, aber ähnlich wie überschattete Properties ist dieser Ansatz mit Vorsicht zu genießen: In kleinen, überschaubaren Projekten kann das durchaus funktionieren. Nimmt die Arbeit größere Dimensionen an, droht Synapsenverhedderungsgefahr – was löst denn der Event noch an Aktionen aus? –, Sie können sich in einer Endlosschleife verfangen, wenn der Event nicht von der auslösenden Subklasse entkoppelt wurde, oder andere Querverkettungen könnten entstehen, die nur schwer zu debuggen sind.

Ach, und außerdem: Die Eventstrukturen funktionieren gut innerhalb der eigenen Klassenhierarchie. Es wird schon einiges aufwendiger, wenn Sie versuchen, darüber unterschiedliche Klassen miteinander reden zu lassen. Das muss manchmal sein, und Sie haben ja schon des öfteren Eventhandler geschrieben. Aber auch hier: Wie bringt eine externe Klasse eine andere dazu, einen Wert abzufragen? Wenn Sie eine Methode, die einen Event raised, von außerhalb zugänglich machen, erinnert das nur noch entfernt an die Prinzipien der objektorientierten Programmierung. Die Klassen sollten so eigenständig wie möglich sein.

Klassensouveränität ist möglich, und darüberhinaus kann man sogar beliebige Klassen zu einer gemeinsamen Art von Superklasse zusammenschmelzen: Mit Interfaces!

Ein **Class Interface** ist in Xojo eine Sammlung von Methodendefinitionen.

Methodendefinitionen kennen Sie gut: Die Festlegung einer Methode mit Namen, Ein- und Rückgabeparametern, aber ohne Programmcode. Also das, was auch beim Definieren eines Events gemacht wird. Nur eben nicht mit Events, sondern wieder normalen Methoden – egal welchen Scope sie haben und ob sie Instanzen- oder Klassen (gesharete) Methoden sind.

 Falls Sie Programmiererfahrung in anderen Sprachen haben: Verwechseln Sie Xojo-Interfaces nicht mit diesen! Objective C und Swift z. B. sehen Interfaces als komplette Methoden-Sammlungen: Nicht nur Methodename und Ein-/Ausgabeparameter gehören dazu, sondern auch der Code der Methoden.

Ich zeige Ihnen mal ein Beispiel aus der Realität; ich glaube, das ist anschaulicher als die abstrakte Menagerie der Vorkapitel.

Mein allererstes Xojo-Programm (vermutlich wird es nie fertig werden) entstand aus dem Frust über schlechten Softwaresupport meines Telefonherstellers. Das fürs Heimbüro extra wegen seines Ethernet-Anschlusses gekaufte Telefon wollte mit dem Mac irgendwann nicht mehr reden. Die Grundlagen des eigenen Programms dafür waren recht schnell fertig, und zum Betatest überredete Leidensgenossen teilten mir bald ihre Misserfolge mit. Ihre Telefone besaßen keinen Ethernetport, sondern kommunizierten via serieller Funkschnittstelle mit dem Rechner. Ansonsten aber auf die gleiche Weise.

Für **Ethernet**-Datenaustausch bietet Xojo das **TCPSocket**, und **Bluetooth** beispielsweise lässt sich über die **Serial**-Klassen ansprechen. Beide Klassen machen ganz ähnliches und besitzen ähnliche Funktionen, funktionieren unter der Haube aber jeweils doch unterschiedlich und gehören keiner gemeinsamen Superklasse an – außer **Object**.

Sicher könnte meine Phone-Klasse, die jeweils ein Telefon verwaltet, jeweils ein TCPsocket und ein Serial beinhalten und zwischen ihnen umschalten. Das würde aber bedeuten, die Kommunikationsstruktur zwischen Phone und Anschluss zweimal zu programmieren: Für beide Anschlussarten. Und dann in der Phone-Klasse jede Menge Abfragen einzubauen, die erst schauen, welcher Anschluss aktiv ist, und dann seine entsprechende Aktion einläuten (Verbindung aufbauen etc.).

Insgesamt also alles andere als modularer und gut zu wartender Code. Sollte noch eine dritte Anschlussmöglichkeit dazukommen, müsste die komplette Phone-Klasse durchforstet und ergänzt werden.

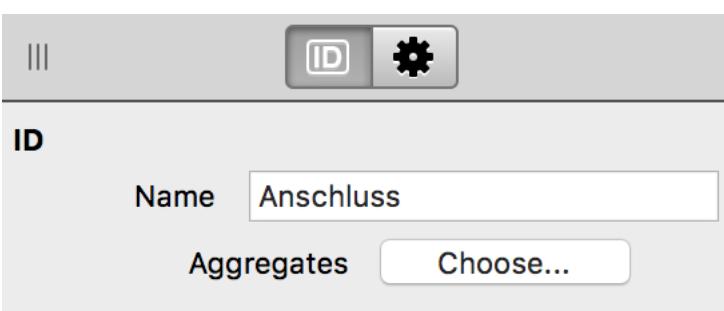
Bei Problemen dieser Art – Wie fasse ich Klassen unter einem Hut zusammen, die aus unterschiedlichen Familien stammen? – helfen Interfaces ungemein.

In diesem Fall bietet es sich an, ein Anschluss-Interface zu entwerfen, das die gängigen Methoden für die Kommunikation über einen Port beinhaltet: Verbindung einrichten, aufbauen, Senden und Empfangen z. B.



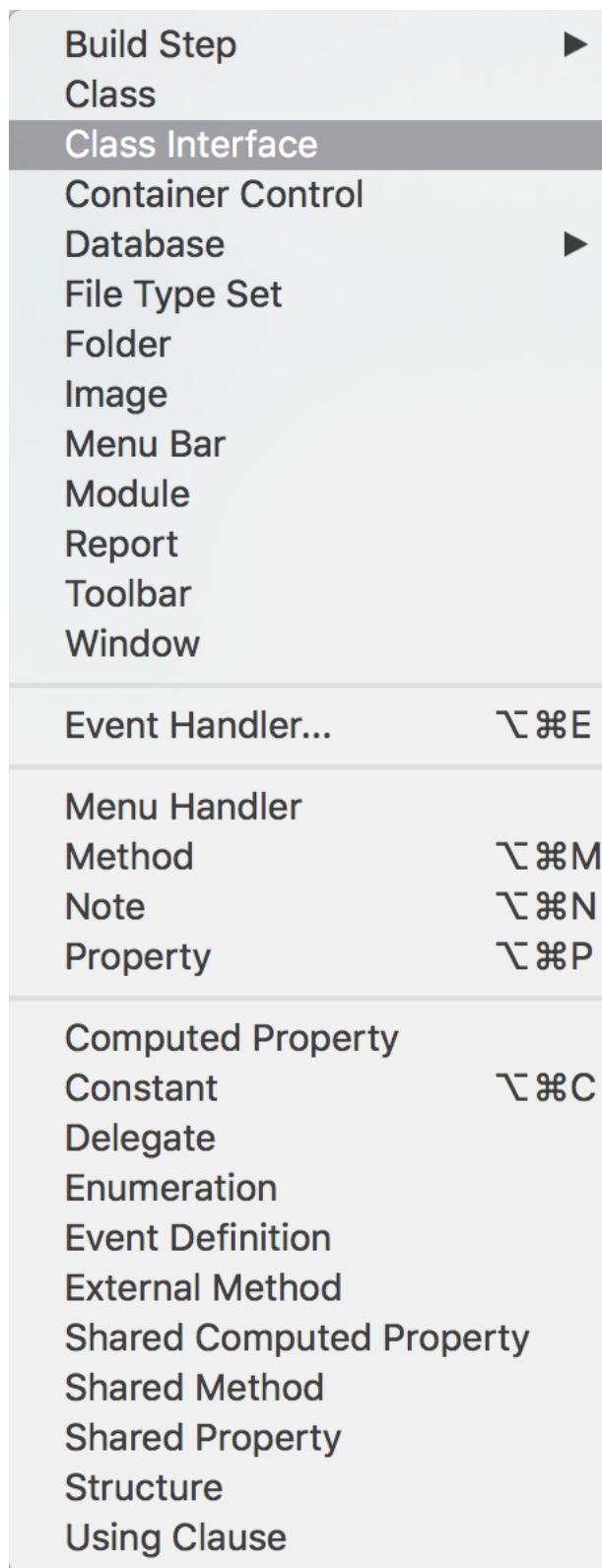
Den folgenden Code müssen Sie jetzt nicht nachbauen, so lange Sie keine Telefonanwendung benötigen. Class Interfaces werden uns noch oft im Beispiel begegnen. Vollziehen Sie den Code aber bitte nach, oder entwerfen Sie ein, zwei eigene Interfaces:

- ▶ Begonnen wird alles mit einem Klick auf das Einfügen-Symbol in der Werkzeugeiste und der Wahl „Class Interface“.
- ▶ Im Inspector kann wie gewohnt ein Name vergeben werden. Ich wähle hier „Anschluss“:

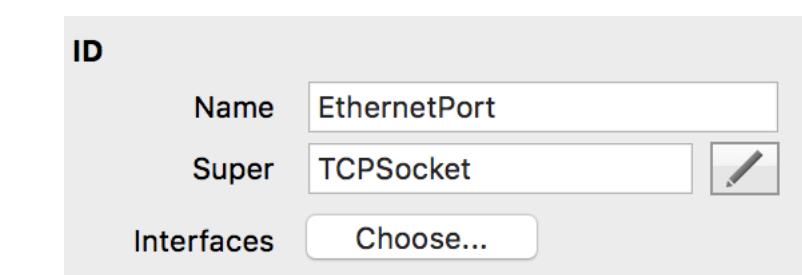


Die Schaltfläche darunter, „Choose...“, ermöglicht, andere Interfaces in dieses neue einzuschließen. Klicken Sie ruhig mal darauf. Beachtliche Liste, oder? Xojo bietet schon sehr viele vorkonfigurierte Interfaces (die von einigen Xojo-Klassen auch benutzt werden). Per Klick auf die Checkbox vor ihren Namen könnte man beliebig viele davon in das Anschluss-Interface integrieren. Deshalb auch das Wörtchen **Aggregates** davor – diese Schaltfläche vereinigt ein Interface mit einem oder mehreren anderern.

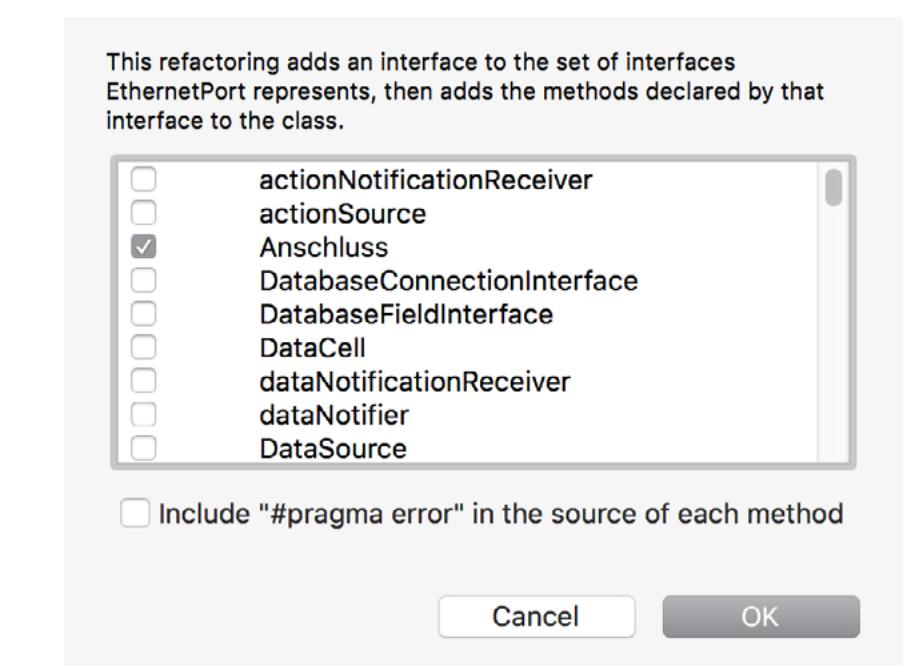
- ▶ Wenn Sie auf das neue Interface im Navigator rechtsklicken, wird Ihnen auffallen, dass die Liste der einfügbaren Dinge sehr überschaubar geworden ist. Methoden und Notizen (Notes) sind möglich, mehr nicht. Zur Demonstration füge ich eine Methode ein und nenne sie „Verbinde“.



- ▶ Weitere Unterschiede zur normalen Klassenmethode werden jetzt offenbar: Der Code-Editor ist deaktiviert. Es lässt sich kein Programmcode eingeben. Wie gesagt, ein Interface sammelt Methodendefinitionen, keinen Methodencode. Ein- und Ausgabeparameter können angegeben werden, bleiben aber in diesem Fall frei.
- ▶ Als nächstes füge ich eine neue Klasse ein, die ich „EthernetPort“ nenne. Ihr Super ist **TCPSocket**.



- ▶ Ein Klick auf „Interfaces/Choose...“ öffnet die von eben schon bekannte Liste. Hier kann nun gewählt werden, welche Interfaces die neue Klasse EthernetPort integrieren soll. Einige sind durch die Superklasse TCPsocket schon als „Inherited“ gekennzeichnet – sie wurden von TCPsocket geerbt. Ich setze noch ein Häkchen bei „Anschluss“.





An dieser Stelle, wo ich nur eine Interface-Methode definiert habe und sie zunächst nur einer Klasse zuordne, ist es unnötig, ein Häkchen bei „**Include "#pragma error" in the source of each method**“ zu setzen. Es kann aber praktisch werden, wenn das Interface umfangreicher ist. Dann wird ein Build zu Fehlermeldungen führen, wenn die via Interface integrierten Methoden noch keinen eigenen Code bekommen haben. Man erlebt also weniger Überraschungen durch Methoden, die irrtümlich gar nichts machen.

- ▶ Nach Bestätigung taucht in der neuen Klasse **EthernetPort** automatisch die **Anschluss**-Methode auf. Ihr Code besteht zunächst aus einer Kommentarzeile:

```
// Part of the Anschluss interface.
```

Es ist ganz sinnvoll, dies so stehen zu lassen, um nicht auf die Idee zu kommen, der individuellen Anschluss-Methode plötzlich Ein- oder Ausgabeparameter zu geben. Das würde sie inkompatibel zum Interface machen, bei dem die Methode ja parameterlos daherkommt.

Hätte ich den Include #prama error-Button beim Integrieren des Interfaces aktiviert, würde hier noch eine weitere Zeile stehen:

```
#pragma error "Don't forget to implement this
method!"
```

Damit würde wie gesagt beim Kompilieren ein Fehler ausgegeben werden, der mich darauf hinweist, dass hier noch individueller Code fehlt.

Der soll jetzt folgen:

```
me.Connect
```

Ziemlich unnötig zu erwähnen: **Connect** ist die Methode, mit der ein Socket versucht, eine Verbindung aufzubauen. Ist das gelungen, feuert das TCPSocket seinen **Connected**-Event.

- ▶ Das Spiel wiederhole ich mit einer weiteren Klasse: **SeriellAnschluss** mit dem Super **Serial**. Auch sie bekommt das Anschluss-Interface.

Hier wird die Realisierung der Verbinde-Methode etwas schwieriger: Serial besitzt die **Open**-Methode, die einen Booleschen Wert als Erfolgsergebnis liefert. Den können wir in der Anschluss-Interfacemethode nicht gebrauchen, weshalb sie in SeriellAnschluss diesen erhält:

```
Dim Ergebnis As Boolean = me.Open
If Ergebnis Then RaiseEvent Connected
```

- ▶ Also bekommt SeriellAnschluss noch eine parameterlose **Connected**-Eventdefinition spendiert, und beide Klassen – EthernetPort und SeriellAnschluss – verhalten sich (fast) identisch, wenn ihre **Verbinde**-Methode aufgerufen wird. Ich muss von einer Klasse, die einer Property ihrer Art beinhaltet, keinen unterschiedlichen Code zum Ansprechen verwenden. Zum Unterschied gleich mehr!

Aber wie kann die Klasse entweder SeriellAnschluss oder EthernetPort beinhalten? Es sind ja beides völlig unterschiedliche Klassen.

Eine Klasse, die ein Interface beinhaltet, wird zugleich zu einer Unterklasse auch des Interfaces.

EthernetPort ist also **sowohl** Unterklasse von TCPSocket **als auch** Unterklasse von Anschluss, und SeriellAnschluss ist **so-wohl** Unterklasse von Serial **als auch** Anschluss-Subklasse.

Um das am Beispiel nachzubauen:

- ▶ Ich lege noch eine neue Klasse an: Telefon. Diese bekommt keine Superklasse, dafür in dieser Demonstration eine Property: Verbindung As Anschluss.

Telefon kann dieser Property jetzt wahlweise einen EthernetPort oder einen SeriellAnschluss zuweisen, etwa

```
Dim Verbindung As New EthernetPort
```

Beide Klassen verhalten sich übrigens noch nicht völlig identisch: **Connected** feuert bei **EthernetPort** auf der **TCPSocket**-Ebene, bei **SeriellAnschluss** auf der Ebene der neuen Klasse. Deshalb muss der Connected-Event bei EthernetPort auf diese Ebene gehievt werden:

- ▶ Ich installiere in EthernetPort einen Connected-Eventhandler, dupliziere den Event mittels „**Create Event Definition from Event**“ und gebe dem Eventhandler den Code

```
RaiseEvent Connected
```

Jetzt verhalten sich beide Klassen beim Verbinden wirklich völlig gleich. Es ist nun möglich, in der Telefon-Klasse den Event auf eine Telefon-Methode zu verbiegen:

```
AddHandler Verbindung.Connected, _
AddressOf Verbunden
```

AddHandler ist ein furchtbar praktischer Befehl. Vielleicht haben Sie sich schon gefragt, wie man an den Connected-Event von Verbindung kommen soll. Die Property liegt ja nicht auf dem Layout, wo man graphisch an die Events gelangen könnte. AddHandler ist das Code-Gegenstück dazu!



Die Syntax von AddHandler ist

AddHandler ((objekt).Event, Delegate)

Delegate ist dabei ein spezieller Datentyp: Ein Verweis auf eine Methode, die hier statt des Events aufgerufen werden soll. Siehe auch „Delegate“ auf Seite 122.

Im obigen Beispiel heißt die Methode Verbunden. Der Delegate zu ihr wird mit dem Befehl **AddressOf** erzeugt.

Wenn Sie bei einem Delegate eine Fehlermeldung erhalten, bezieht sich diese häufig darauf, dass andere Parameter erwartet wurden als gegeben sind. Machen Sie sich daher einen Knoten ins Taschentuch:

Ein Delegate erwartet als ersten Parameter immer das aufrufende Objekt!

AddHandler wird häufig verwendet, um den Action-Event eines programmatisch erzeugten Timers auf eine Methode zu verbinden. Wenn Sie sich die Xojo-Timer-Beispiele anschauen, sehen Sie, dass diese Methoden als ersten Parameter stets den Timer selbst entgegennehmen. Deshalb muss die **Verbunden**-Methode auch den Eingabe-Parameter

Port As Anschluss

erwarten.

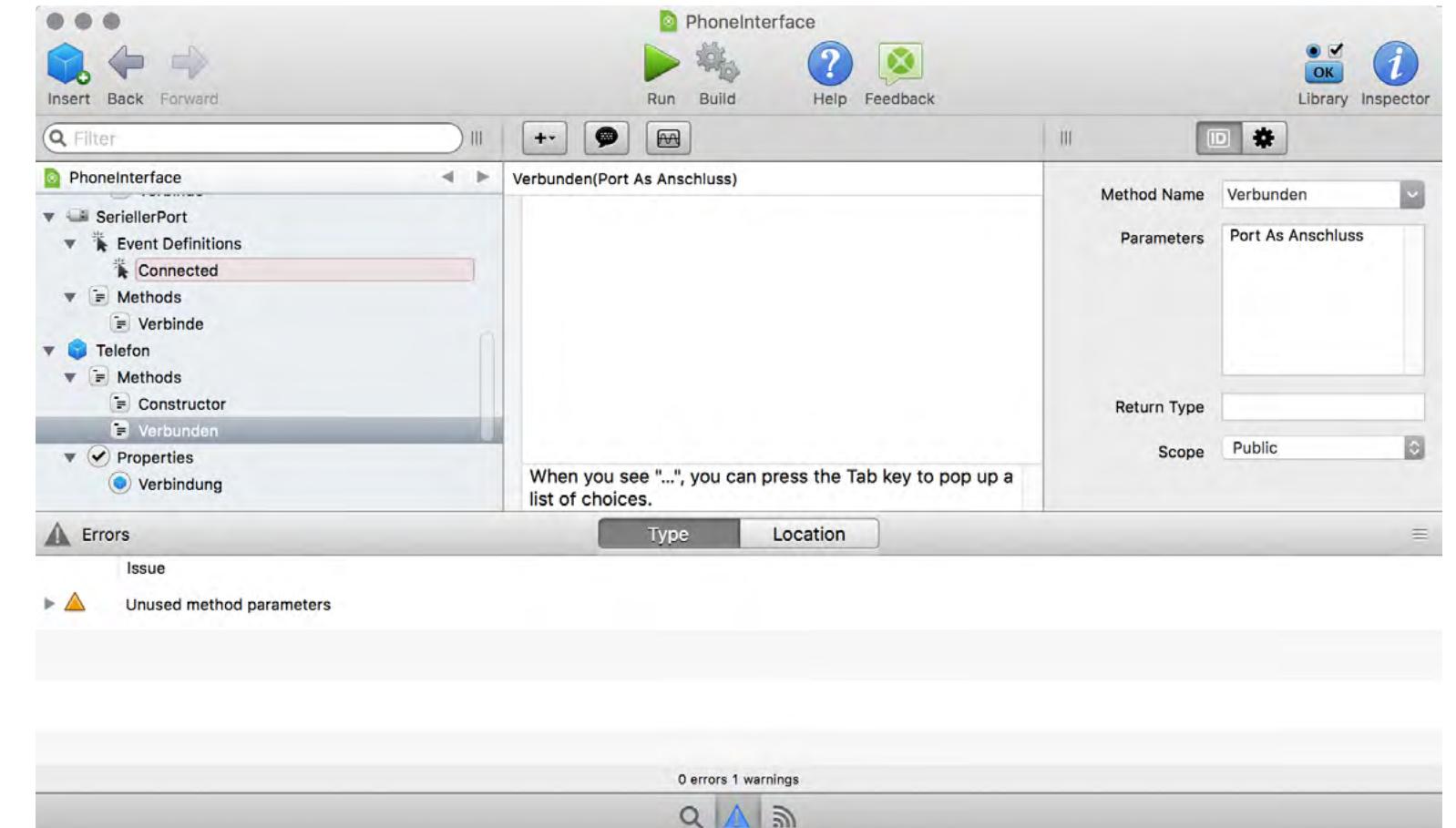
Was bei Timer-Delegates also manchmal etwas nervig ist:
Man maskiert den erhaltenen Timer t meist mit
`#pragma unused t`
aus, weil man ihn gar nicht braucht, ist also eigentlich wahnsinnig praktisch: Die Telefon-Klasse muss in ihrer

Verbunden-Methode nicht umständlich prüfen, welche Anschlussart vorliegt, sondern bekommt den Anschluss gleich frei Haus geliefert.

AddHandler besitzt übrigens ein Gegenstück, **RemoveHandler**. Damit wird ein per AddHandler verbogener Event wieder zurückgesetzt. Dies **muss erfolgen**, falls Sie dem Event einen neuen Delegate zuweisen wollen. Andernfalls wird Ihr Programm an dieser Stelle die Zusammenarbeit einstellen. Und nebenbei ist es guter Programmierstil, beim Beenden alles wieder zurückzusetzen – und mag in einigen Fällen **Querverreferenzen** auflösen, die ansonsten dafür sorgen, dass einige nicht mehr benötigte Objekte nicht im Speicher freigegeben werden – die beürchtigten **Memory Leaks**.

Als Einführung zu den Interfaces mag dieses Fragment erst einmal genügen. Die Ports sind unterschiedlich zu konfigurieren, und solange Sie selbst kein eigenes Programm in dieser Art planen, will ich Sie nicht mit an dieser Stelle unnötigen Details plagen. Falls aber doch, finden Sie das kleine Minimalprojekt als Startrahmen für eigene Experimente.

Letzte Anmerkung in diesem Abschnitt: Ein Class Interface kann auch **ganz ohne Methodendefinitionen** daherkommen. Dies kann ganz praktisch sein, wenn man nur bestimmte Klassen als Parameter zulassen will, oder um Klassen verschiedener Frameworks als Parameter zuzulassen und interne Konvertierungen vorzunehmen. Aber das ist eine andere Geschichte und soll ein andermal erzählt werden ...





3.8.1. Eilnachricht für ...

Class Interfaces haben noch einen anderen, sehr, sehr praktischen Nutzen. Nun gut, sie haben noch viele Nutzen, aber über folgendes Problem stolpert programmierer regelmäßig einmal:

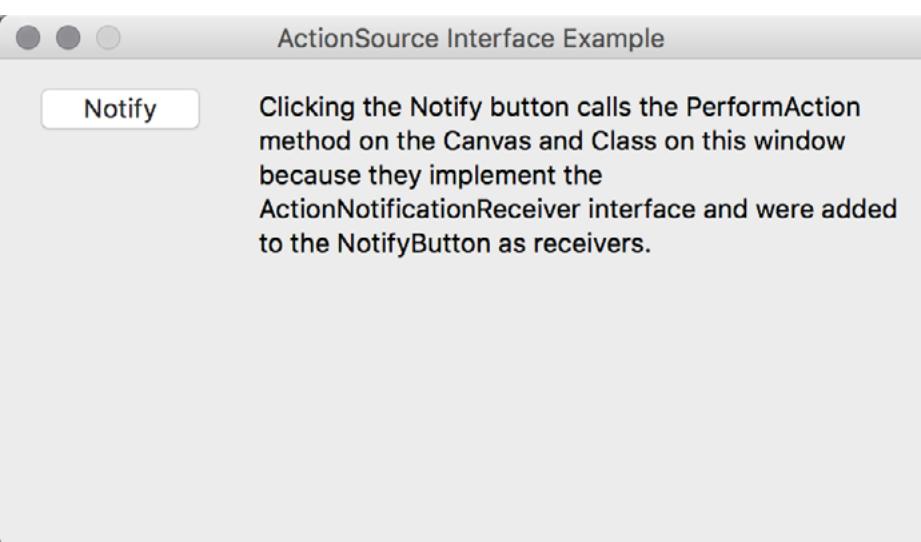
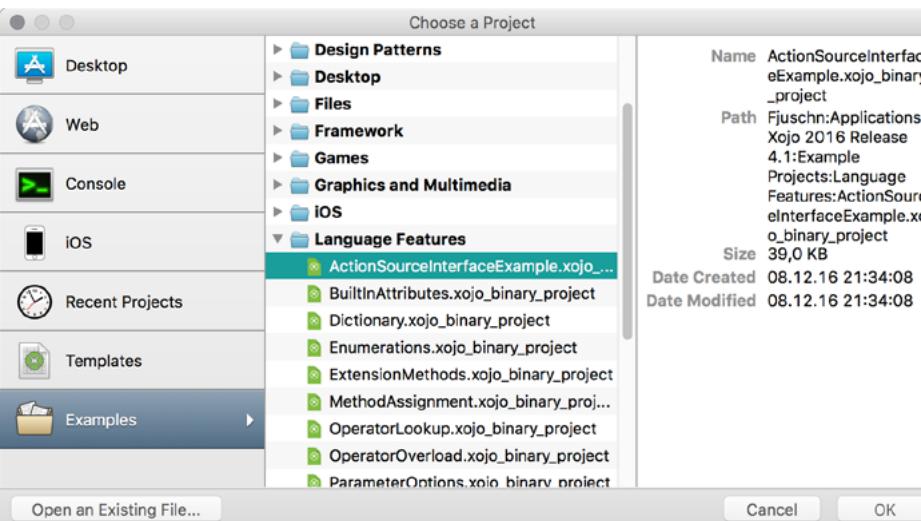
Irgendeine Klasse soll mit irgendeiner anderen kommunizieren. Womöglich sogar mit mehreren (quasi) gleichzeitig. Um zu einem halbabstrakten Beispiel zu greifen:

- ▶ Öffnen Sie bitte das Beispielprojekt „ActionSourceInterfaceExample“ aus Xojos Examples-Liste.
- ▶ Klicken Sie auf „Run“ und drücken Sie dann auf den Button „Notify“ im Programmfenster.

Erst mal wenig spektakulät, oder? Eine Messagebox erscheint und ein zusätzlicher Text wird auf dem Fenster eingeblendet. Erst mal gar nichts besonderes, nur: Schauen Sie sich einmal den **Notifybutton** von **Window1** im Navigator an. Finden Sie seinen Action-Eventhandler? Können Sie nicht, denn es gibt keinen. Sie könnten einen hinzufügen. Trotzdem reagieren **WinCanvasReceiver**, die Instanz der **CanvasReceiver-Klasse**, die im Projekt als Subklasse von **Canvas** angelegt wurde, dem Standard-Desktop-Steuerelement für individuelle Text- und Grafikausgaben und **WinClassReceiver**, die völlig eigenständige Klasse, die nur eine MsgBox ausgeben kann.

Die Lösung dafür finden Sie in der Definition der beiden letzten Klassen: Beide benutzen das **actionNotificationReceiver**-Interface, ein von Xojo schon vordefiniertes Interface für Klassen, die auf einen einfachen Action-Befehl (also ohne weitere Parameter) reagieren sollen.

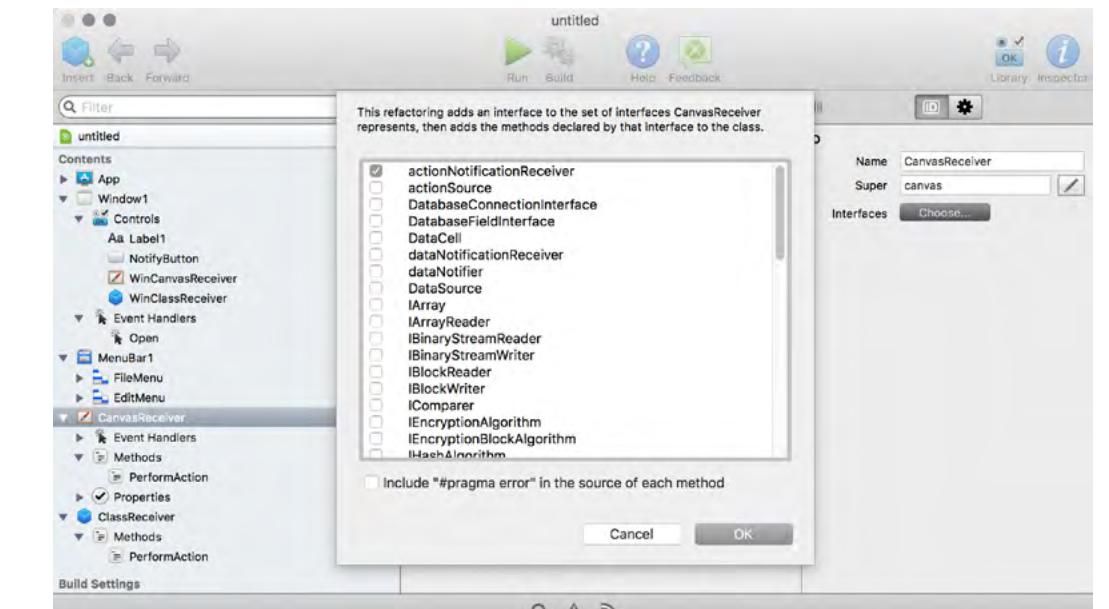
Dieses Interface besitzt nur eine Methode: **PerformAction**. Da es sich um eine Interface-Methode handelt, ist ihr Code in-



dividuell in den Klassen oder Instanzen definierbar. Beim **Class-Receiver** ist dies

```
Msgbox("Notification received by ClassReceiver.")
```

und **CanvasReceiver** setzt eine interne Text-Property und ruft dann via **Invalidate** seinen **Paint**-Befehl auf, um den Text auf sich selbst zu „malen“. (Das sind Details, die im Desktop-Kapitel genauer beschrieben werden.)



Dass die beiden Controls auf den Knopfdruck anspringen, verdanken sie dem Code im **Open**-Event von **Window1**. Dort werden sie beim Button angemeldet:

```
NotifyButton.AddActionNotificationReceiver _  
    (WinClassReceiver)  
NotifyButton.AddActionNotificationReceiver _  
    (WinCanvasReceiver)
```

AddActionNotificationReceiver schließlich macht genau das, was der Name vermuten lässt: Es fügt ein Objekt, das das **actionNotificationReceiver**-Interface benutzt, zur Liste von Objekten hinzu, die im Falle des Action-Events alarmiert werden sollen, indem die Interface-Methode **PerformAction** aufgerufen wird.

AddActionNotificationReceiver ist übrigens seinerseits Teil eines Interfaces namens **actionSource**. Praktischerweise ge-



hört das zur Standardausstattung diverse Steuerelemente, die über einen Action-Event verfügen. Sie müssen außer An- und Abmeldungen der Empfänger also nichts weiter tun, um dynamisch Empfänger benachrichtigen zu lassen.

Und wo wir gerade vom Abmelden sprechen: Die dafür zuständige Methode, die das **actionSource**-Interface bietet, lautet

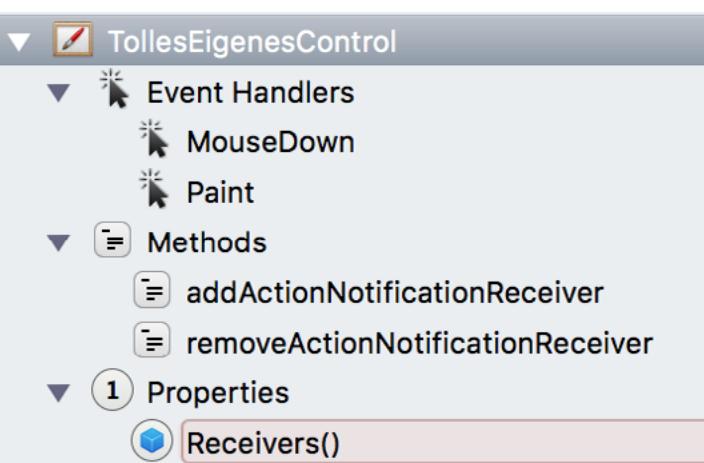
RemoveActionNotificationReceiver –
(Receiver As ActionNotificationReceiver)

3.8.2. ... eigene Klassen?

Nehmen wir mal an, Sie haben ein supertolles ganz eigenes Steuerelement gebaut. Das macht man in der Regel mit einem **Canvas**, dem noch zu beschreibenden voll maßschneiderbaren Grafik-Steuerelement. Canvas kennt aber keinen **Action**-Event und benutzt deshalb auch nicht das **actionSource**-Interface.

Lässt sich aber alles beheben! Canvas kennt dafür einen **MouseDown**-Event – eine Maustaste wurde gedrückt.

Das ganze mal im Schweinsgalopp – ich denke, ich muss Ihnen nicht mehr für jeden Schritt die Hand führen, oder? In der Abbildung rechts meine Canvas-Unterklasse „TollesEigenesControl“, die nur ein abgerundetes Rechteck mal und bei Klick auf ihre Fläche angemeldete NotificationReceiver informiert. Und folgend einfach der Code, jeweils mit seinem kompletten Methodenkopf, damit Sie wissen, was was ist:



```
Function MouseDown(X As Integer, Y As Integer) _  
Handles MouseDown as Boolean  
For Each Empfänger As actionNotificationReceiver _  
in Receivers  
    Empfänger.PerformAction  
Next  
End Function
```

```
Sub Paint(g As Graphics, areas() As _  
REALbasic.Rect) Handles Paint  
    g.ForeColor = Color.LightGray  
    g.FillRoundRect (0,0,g.Width, g.Height, 4, 4)  
End Sub
```

```
Public Sub addActionNotificationReceiver _  
(receiver As actionNotificationReceiver)  
// Part of the actionSource interface.  
me.Receivers.Append receiver  
End Sub
```

```
Public Sub removeActionNotificationReceiver _  
(receiver As actionNotificationReceiver)  
// Part of the actionSource interface.  
Dim p as integer = receivers.IndexOf(receiver)  
If p > -1 then receivers.Remove (p)  
End Sub
```

Receivers() schließlich ist ein Array des Typs **actionNotificationReceiver**.

Der Reihe nach:

Im **MouseDown**-Eventhandler wird einfach bei jedem angemeldeten Receiver (allen denen, die via **addActionNotificationReceiver** an das Array-Property Receivers() angehängt wurden, die PerformAction-Methode aufgerufen.

Neu ist hier die Schleife **For ... Each**. Damit lässt sich eleganter über ein Array iterieren (also jedes Element einmal aufrufen) als mit dem Äquivalent

```
For i As Integer = 0 to Receivers.UBound -1  
    Dim Receiver As ActionNotificationReceiver = _  
        Receivers(i)  
    Receiver.PerformAction  
Next
```

Wobei **For ... Each** nicht unbedingt die Reihenfolge des Arrays einhält, was in diesem Fall völlig wurst ist. Auch vegane Wurst.

Der Paint-Event zeichnet einfach ein abgerundetes graues Rechteck.

addActionNotificationReceiver hängt einen NotificationReceiver an das Array **Receivers**, und **removeActionNotifi-**



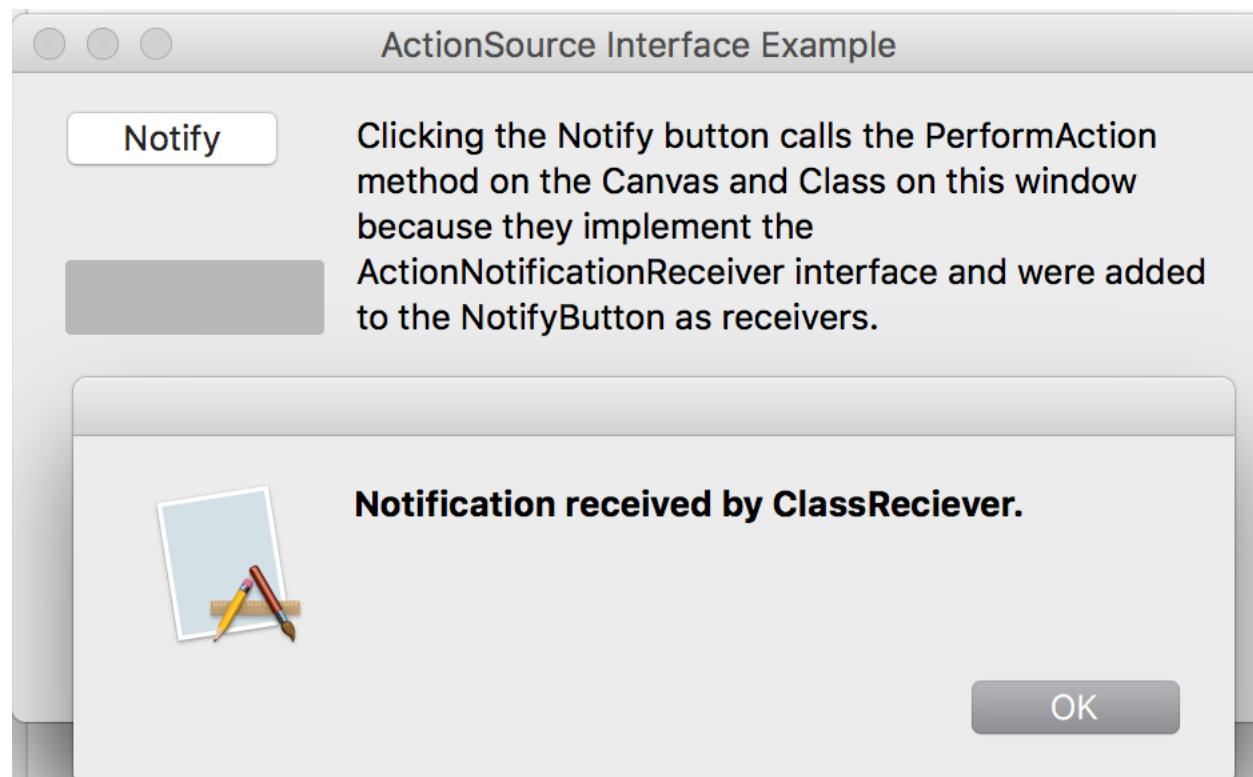
actionReceiver schaut nach, ob der angegebene Receiver Bestandteil des Arrays ist. Falls ja, wird er daraus gestrichen.

Jetzt einfach beherzt eine Instanz von TollesEigenesControl aus der Library auf das Window1-Layout gezogen und den Open-Event mit

```
TollesEigenesControl1.AddActionNotificationReceiver _  
    (WinClassReceiver)  
TollesEigenesControl1.AddActionNotificationReceiver _  
    (WinCanvasReceiver)
```

ergänzt und fortan ist es egal, ob Sie auf den Button oder das eigene (eigentlich nich nicht ganz so tolle) Control anklicken: Beide kommunizieren gleichberechtigt mit den anderen Steuerelementen.

 **zum schnellen Nachvollziehen** finden Sie dieses Projekt auch wieder bei den Beispielprojekten. Inkl. Original-Tippfehler!





3.9. Stunde der Entscheidung

Ich habe (hoffentlich) wieder einmal eine Überraschung für Sie:
Wir sind fast durch mit den wichtigsten Grundlagen zu OOP!
Es gibt allerdings noch einen letzten wichtigen Unterschied der Objekte zu den elementaren Datentypen. Um diesen wirklich zu verstehen, muss man sich allerdings mit den Objekten auf Rechnerebene auseinandersetzen. Das mag an dieser Stelle viel mehr sein, als Sie gerade ein detailliert wissen wollen. Ich werde daher die Eigenarten am Anfang zusammenfassen, und das Warum folgt dann in Form von Fortgeschrittenen-Infos direkt im Anschluss.

Mein Tipp wäre: Wird Ihnen die Erklärung zu viel, dann überspringen Sie sie erst einmal. Wenn Sie dann ein Stück mehr praktische Erfahrung gesammelt haben, sollten Sie sie aber baldigst nachholen. Es ist, das können Sie sich denken, ein Häppchen Schlüsselwissen, das nicht nur den Einblick in Xojo, sondern so ziemlich in alle objektorientierten Systeme kräftig nachkontrastiert.



3.10. Wenn gleich nicht gleich gleich ist

Lassen Sie uns zum Start mal ein frisches Kommandozeilen-Projekt anlegen und den Run-Eventhandler mit diesem einfachen Code füllen:

```
Dim ErsteZahl As Integer = 1
Dim ZweiteZahl As Integer = ErsteZahl
ZweiteZahl = ZweiteZahl + 1
Print ErsteZahl.ToString+", "+ZweiteZahl.ToString
```

Dieses Programm müssen wir nicht groß diskutieren, oder? Wenn Sie es ausführen lassen, erhalten Sie erwartungsgemäß das Ergebnis

1, 2

Ergänzen Sie den Quellcode mit den folgenden Zeilen:

```
Using Xojo.Core
Dim ErstesDatum As Date = Date.Now
Dim ZweitesDatum As Date = ErstesDatum
ZweitesDatum = ZweitesDatum + New -
    DateInterval(0,0,1)
Print ErstesDatum.ToString+", " + -
    ZweitesDatum.ToString
```

Hier gibt es mal wieder eine neue Vokabel, nämlich

Using

Using – „Verwendend“ – ist wieder mal so eine hübsche Tipp-Ersparnis. Mit Using weisen Sie Xojo an, bevorzugt Klassen und Funktionen des angegebenen Frameworks zu benutzen.

Sie wissen ja, dass es eine alte Datumsklasse gibt – eben Date –, ich Ihnen aber bevorzugt das neue Xojo-Framework zeigen möchte. Um nun nicht jedesmal Xojo.Core.Date und Xojo.Core.DateInterval tippen zu müssen, verwende ich

Using Xojo.Core,

woraufhin sich Xojo im Konfliktfall ein Xojo.Core. da-zudenkt. Statt der alten Date-Klasse zugehörig zu sein, werden ErstesDatum und ZweitesDatum Instanzen der Klasse Xojo.Core.Date.

Das gilt übrigens auch für ganze Klassen! Wenn Sie mit Rechts auf das App-Objekt klicken (oder einen der anderen beschriebenen Einfügewege wählen), dann sehen Sie, dass Sie auch eine Verwendungsklausel – eine

Using Clause

einfügen können. In den Inspector schreiben Sie dann einfach das gewünschte Framework – hier also Xojo.Core –, und die ganze Klasse bzw. das ganze Objekt präferiert fortan das Xojo-Framework. Im Navigator wird dieser Umstand auch angezeigt – siehe rechts. Müssen Sie jetzt aber nicht so einfügen, einmal Using reicht völlig!

The screenshot shows the Xojo Inspector interface. A context menu is open over an 'App' object, with 'Using Clause' highlighted in grey. The menu includes options like 'Add to "App"', 'Inspect "App"', 'Open in New Tab', 'Cut', 'Copy', 'Paste', 'Delete', 'Duplicate', 'Find 'App'', 'Make External...', 'Encrypt "App" ...', 'Export "App"...', 'Print "App"...', 'Extract Interface...', 'Extract Superclass...', 'Implement Interface', and 'Inspector Behavior...'. Below the menu, the 'Using Clauses' section of the Inspector is visible, showing a tree structure with 'Using Xojo.Core' expanded.



Auch hier: Führen Sie das Programm einmal aus.

Nach 1, 2 erscheint eine Ausgabezeile, die den aktuellen Tag und den morgigen ausgibt, gell? Also irgendwie in der Form

```
2016-09-01 12:30:03, 2016-09-02 12:30:03
```

Alles noch wenig überraschend, oder? Dann erweitern Sie das Programm bitte noch ein Stückchen, und zwar so:

```
Dim ErstesAltesDatum As new Global.Date
Dim ZweitesAltesDatum As Global.Date = ¬
    ErstesAltesDatum
ZweitesAltesDatum.day = ¬
    ZweitesAltesDatum.Day + 1
Print ErstesAltesDatum.SQLDateTime+ ", " + ¬
    ZweitesAltesDatum.SQLDateTime
```

Das ist eigentlich genau der Code, der dem vorigen Abschnitt entspricht, nur für das alte Framework. Damit wir dieses auch innerhalb der Using Clause verwenden können (Sie erinnern sich? Durch Using Xojo.Core entspricht Date ja der Xojo.Core.Date-Klasse), gebe ich das richtige Framework explizit an – und das hört bei Xojo auf den Namen

Global

SQLDateTime

wiederum ist eine Property der Global.Date-Klasse, die als String existiert und dem Standardformat entspricht, das auch Xojo.Core.Date.ToText erzeugt, wenn dort keine weiteren Formatierungsparameter übergeben werden – mit anderen Worten: Die Ausgabe sieht genau aus wie im Beispiel oben.

Und wenn Sie nun wieder auf Run klicken, dann könnte Sie das Ergebnis verblüffen:

```
1, 2
2016-09-01 12:30:03, 2016-09-02 12:30:03
2016-09-02 12:30:03, 2016-09-02 12:30:03
```

Wir haben nur ZweitesAltesDatum verändert, aber ErstesAltesDatum hat diesen Wert auch angenommen – es liegt einen Tag in der Zukunft.

Ist Xojo kaputt?

Blöde Rhetorik, ich weiß. Wenn's so wäre, würde ich Ihnen diesen Umstand kaum so lang und breit präsentieren. Wir haben hier vielmehr ein Feature der objektorientierten Programmierung vor uns: Die **Parameterübertragung**

per Referenz

oder, wie es in Xojo und auch andernorts heißt,

by Reference bzw. kurz byRef

Wenn es ein Fachwort gibt, dann existiert häufig auch ein Fachbegriff für die Dinge, die ihm nicht entsprechen. Und das gibt es in diesem Fall und ist uns wohlbekannt, wenn wir's auch noch nicht angesprochen haben: Die Parameterübertragung

per Wert

bzw.

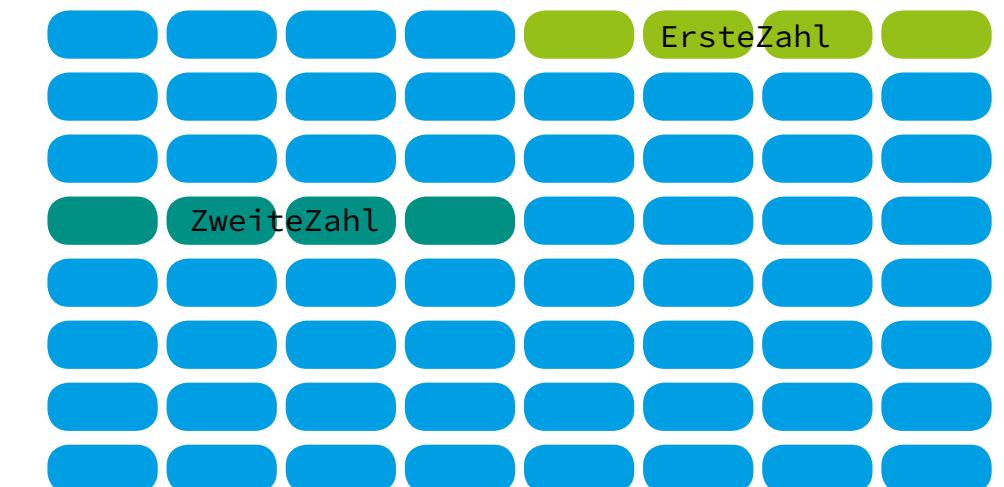
by Value oder byVal.

Diese Form der Variablenübergabe kennen Sie von den einfachen Datentypen wie den Integers des ersten Programmabschnitts. Dort repräsentiert der reservierte Speicherbereich zugleich den Wert. ErsteZahl steht also irgendwo im Speicher und hat den Wert 1 (hier der Bequemlichkeit halber nur als 8 Bit dargestellt). Sie wissen ja: In der Realität hat ein Integer 32 Bit auf einem 32 Bit-System und 64 Bit auf einem 64 Bit-OS:

```
27 26 25 24 23 22 21 20
```

Und an irgendeiner anderen Stelle im Speicher wird ZweiteZahl angelegt, erhält den Wert von ErsteZahl und noch 1 obendrauf, sieht also am letzten Byte so aus:

```
27 26 25 24 23 22 21 20
```



Hier zur Übersicht als 32 Bit-Integers, ein Kästchen entspricht also einem Byte: Zwei individuelle Speicherbereiche mit individuellen Werten. Ändern Sie ErsteZahl, so hat dies keine Auswirkungen auf ZweiteZahl und umgekehrt.

Das gleiche Übertragungsprinzip per Wert gilt auch bei den Klassen, die im Prinzip einfachen Datentypen mit ein bisschen was drumherum entsprechen. So mag der Kern eines Xojo.Core.Date-



Objekts im Prinzip die Double-Property `SecondsFrom1970` sein. Die restlichen Properties und Methoden könnten einfach Computed Properties sein, die aus diesem eigentlichen Wert die gewünschten Eigenschaften berechnen, bzw. Methoden, die ja ohnehin nur einmal im Speicher auf Klassenebene existieren.

Das Global.Date-Objekt basiert allerdings auf einem „richtigen“ Objekt, was ein Blick in die [Sprachreferenz](#) offenbart:

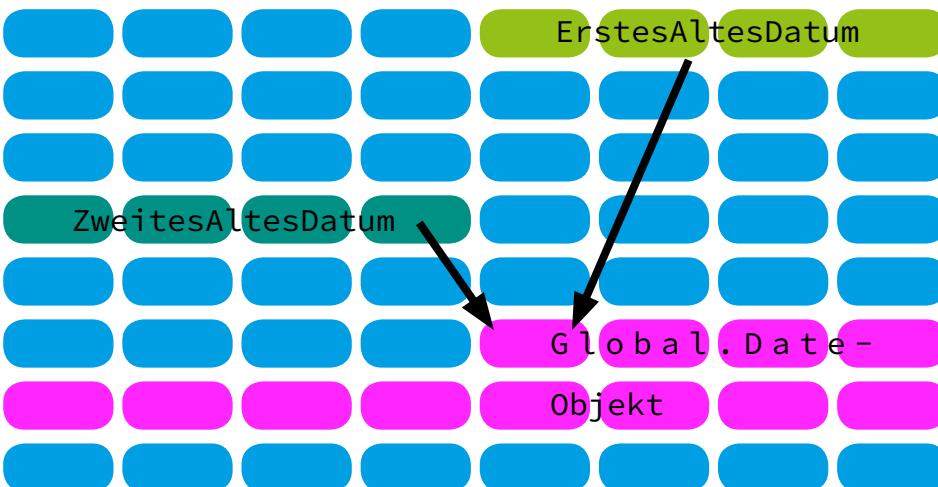
Date _____ Class _____

Class (inherits from Object)

Ein Objekt mag sehr viele Properties beinhalten, was bedeuten würde, bei jeder Parameterübergabe eines Objekts auch sämtliche Properties in einen neuen, größeren Speicherbereich hinüberzukopieren. Egal wie schnell Ihr Rechner ist: Speicher ist immer noch einer der Flaschenhälse bei der Programmiererei, weshalb man größere Kopieraktionen nach Möglichkeit vermeidet oder, wo sie alternativlos sind, etwa bei der Berechnung von Spielgrafiken, lieber der damit viel besser klarkommenden Grafikkarte anvertraut.

Vom Zeitfaktor einmal abgesehen: Handelt es sich beim Objekt gar um ein grafisches oder sonstwie an die Hardware oder Physik gebundenes wie ein `HTTPSocket`, über das Sie Internetdaten empfangen können, dann wollen Sie ja auch gar keine Kopie, sondern Zugriff auf das Original!

Und aus ebenjenen Gründen werden Objekte in Xojo (wie in vielen anderen Sprachen auch) nicht per Wert transferiert, sondern per Referenz: Die Variablen `ErstesAltesDatum` und `ZweitesAltesDatum` sind nichts weiter als Verweise auf den Speicherbereich, in dem das eigentliche `Global.Date`-Objekt angelegt wurde. Oder im Schaubild:



Auf diese Art der Parameterübergabe ist ein Objekt genauso schnell übergeben wie ein einfacher Integer: Im Grunde wird nur die Speicheradresse des eigentlichen Objekts kopiert. Was ganz anschaulich machen müsste, warum nun im letzten Programmabschnitt beide `AlteDaten` den gleichen Wert tragen: Mit der Zeile

```
Dim ZweitesAltesDatum As Global.Date = -  
    ErstesAltesDatum
```

haben wir den „Zeiger“ von `ZweitesAltesDatum` auf den gleichen Wert gesetzt, den `ErstesAltesDatum` besitzt. Damit adressieren beide `AlteDaten` den gleichen Objekt-Speicherbereich, und somit wirkt sich jede Veränderung an einer der Variablen auch auf die andere aus, weil sie im Grunde identisch sind.

 Ich weiß, diesen Hinweis kann ich mir sparen, aber trotzdem: Machen Sie sich keine Gedanken, wie Sie vermeiden können, über die Unterschiede der Parameter-Übergabe zu stolpern. Früher oder später passiert das jedem, und meistens mehr als einmal. Jetzt können Sie aber nicht mehr sagen, Sie wären nicht gewarnt worden!

Ach, und da wir schon über Zeiger sprachen: Ja, diese existieren als einfacher Datentyp in Xojo und hören auf den Namen

P+

– also kurz für Pointer, auf Deutsch Zeiger.

Sie repräsentieren eine Speicheradresse, und als solche nehmen sie wie Integers 32 Bit auf 32 Bit-Systemen und 64 Bit auf 64 Bit-Systemen ein. Technisch gesehen sind sie nichts weiter als ein Integer, der aber eben explizit eine Speicheradresse darstellen soll.

 Als kleiner Vorweggriff: Bei vielen Objekten in Xojo können Sie die Speicheradresse des eigentlichen Objekts bequem einsehen, und zwar über die Handle-Property (wobei nicht alle Objekte diese Property offenbaren). Die Austauschbarkeit von `Ptr` und `Integer` können Sie dabei auch gleich erleben: Unter iOS sind die Handles stringent als Properties des Typs `Ptr` angelegt, während die Desktop-Systeme dafür (wohl aus historischen und Kompatibilitätsgründen) einen `Integer`-Wert verwenden.

3.10.1. Keine Regel ohne Ausnahme – oder?

Was, wenn Sie doch einmal eine Kopie eines Objekts benötigen? Entweder haben Sie Glück, und die Objektklasse bietet eine `Clone`-Methode, oder Sie müssen die Ärmel hochkrempeisen und sich selbst so etwas schreiben – also ein neues Objekt anlegen und die relevanten Properties manuell kopieren.



In die andere Richtung allerdings gibt es eine Lösung.

Nehmen wir mal an, Sie schreiben ein mathematisch anspruchsvolleres Programm, und eine Ihrer Methoden verwendet mehrere Eingabeparameter als einfache Datentypen, von denen einige dabei auch verändert werden und mit den neuen Werten für die weitere Berechnung in der Hauptmethode zur Verfügung stehen sollen. Sowas wie

```
Public Function KomplexeBerechnung (a As Integer, b As Integer, c As Integer) As Boolean
If a < b Then c = b * a
If c/a <= b * a Then b = b + b * a - c
If a + b + c < 20000 then Return True
End Function
```

Übergeben Sie dieser Funktion (fragen Sie mich nicht nach Sinn und Zweck, bitte!) jetzt drei Integer-Parameter, können Sie zwar den Booleschen Ergebniswert erhalten, aber die drei Übergabe-Integers sind außerhalb der Methode unverändert. Ja, man könnte Sie in eine andere Struktur verpacken (die wir hier noch nicht besprochen haben), diese zurückgeben und auf der anderen Seite wieder alles extrahieren. Das kostet Zeit und Nerven, und es wäre doch viel leichter, wenn man sich diese Entwicklungszeit sparen könnte, oder?

Kann man auch! Nämlich einfach durch Ergänzung des Schlüsselworts `ByRef` vor den Übergabeparametern, also

```
Public Function KomplexeBerechnung (ByRef a As Integer, ByRef b As Integer, Byref c As Integer) As Boolean
```

Und rufen Sie das ganze jetzt auf, z.B. via

```
Dim ErsteZahl As Integer = 1
Dim ZweiteZahl As Integer = ErsteZahl + 1
Dim Drittezahl as integer = 12
Dim Resultat as Boolean = KomplexeBerechnung (ErsteZahl, ZweiteZahl, Drittezahl)
Print ErsteZahl.ToString + ", " + ZweiteZahl.ToString + ", " + Drittezahl.ToString
```

dann erhalten Sie

```
1, 2, 2
```

als Ausgabe. Die Parameter wurden als Referenz übergeben und haben damit die Original-Variablen modifiziert – zumindest `Drittezahl` bei den obigen Werten.

3.10.2. Objekt-Apologetik

Ich habe bisher fein zwischen Datentypen und Objekten unterschieden, so wie Xojo sich das auch zueigen gemacht hat. Wenn wir aber mal ganz genau sind, dann steckt zumindest hinter der Objekt-Variablen ja auch ein ganz primitiver Datentyp, der Pointer. Insofern, und nicht zuletzt auch, weil jedes Objekt freilich auch nur aus Daten besteht: Ja, strenggenommen sind Objekte auch eine Form von Datentyp. Wenn Sie sich die wie üblich leider viel zu akademische [Definition bei Wikipedia](#) anschauen, sehen Sie diesen Verdacht auch bestätigt: Das, was ich bislang als Datentyp bezeichnete, wird dort als

elementarer oder primitiver Datentyp

betitelt, und die Objekte laufen dort als

Zeigertyp oder dynamischer Datentyp.

Also nur, damit Sie nicht in Streitgespräche ob der Natur von Objekten geraten. Jedenfalls außerhalb Ihres Philosophiezirkels. Ich bleibe aber auch weiterhin bei der vereinfachten Unterscheidung von Datentypen und Objekten – ich finde, man denkt so um weniger Ecken. Einverstanden?



3.10.3. Haben Sie irgendwelche Referenzen?

Wenn Sie sich noch einmal das Schema von Seite 97 anschauen: Wie ist es dann um die Lebenszeit des eigentlichen Objekts bestimmt, also des reservierten Speicherbereichs, das es einnimmt? Bei den einfachen Datentypen, wo Speicherbereich = Wert gilt, war der Scope ja einfach zu bestimmen, da er identisch mit dem Lebenszyklus der Variable ist.

Aber beim Objekt? Lebt das Datum so lange wie ErstesAltesDatum? Wie ZweitesAltesDatum? Wann weiß der Computer, dass es sicher ist, seinen Speicher wieder freizugeben?

In früheren Zeiten war das in der Tat nicht trivial: Programmierer mussten sich in diversen Programmiersprachen und Betriebssystemen selbst um die Speicherverwaltung kümmern, etwa indem sie manuell den belegten Speicher wieder freigaben oder das nicht mehr benötigte Objekt als frei verfügbar markierten und dann die [Müllabfuhr](#) beauftragten.

Heute vertraut man in vielen Programmiersprachen und Betriebssystemen dagegen einer Technik namens

ARC oder Automatic Reference Counting

oder aber auch auf Deutsch

(Automatische) Referenzzählung

Und, na klar, Xojo macht das ebenso.

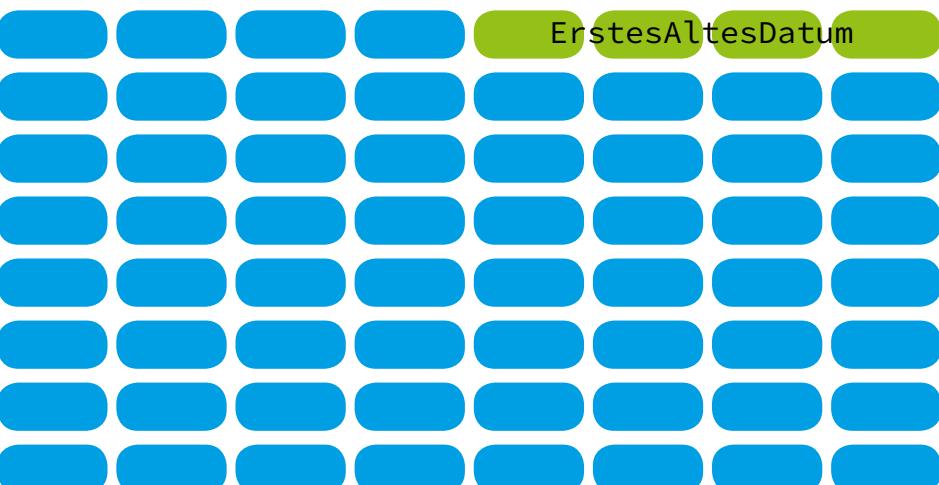
Das Prinzip ist, bei aller Orangenheit dieser Seite, überhaupt nicht schwer: Jedes Objekt besitzt einen (unsichtbaren) Zähler. Der wird um 1 erhöht, sobald eine Variable darauf verweist. Und

verschwindet eine solche Variable wieder aus dem Scope, geht's entsprechend um 1 runter.

Hat der Referenzzähler den Wert 0, dann weiß das System, dass keine Variable mehr auf den Objektspeicher verweist. Bei der nächsten Gelegenheit wird der belegte Speicher dann wieder freigegeben.

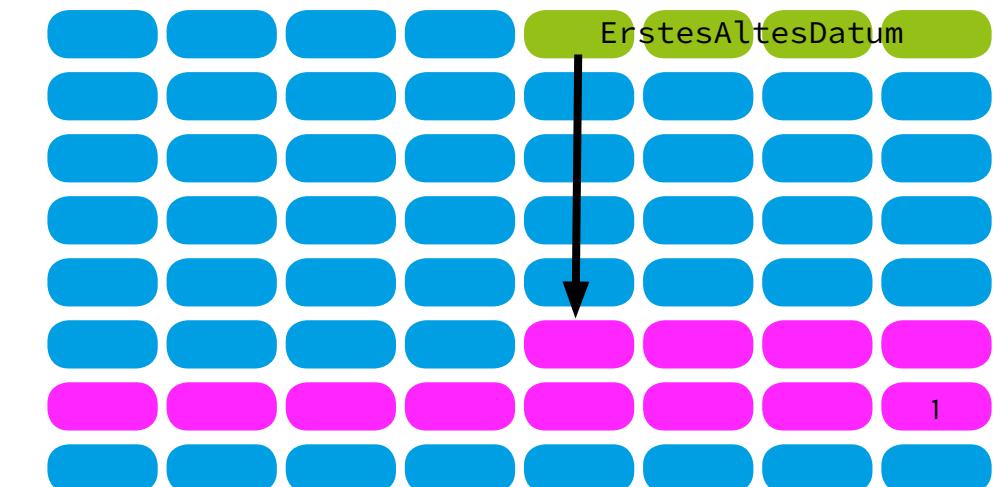
Zum Nachvollziehen hier als Daumenkino:

```
Dim ErstesAltesDatum As Global.Date
```



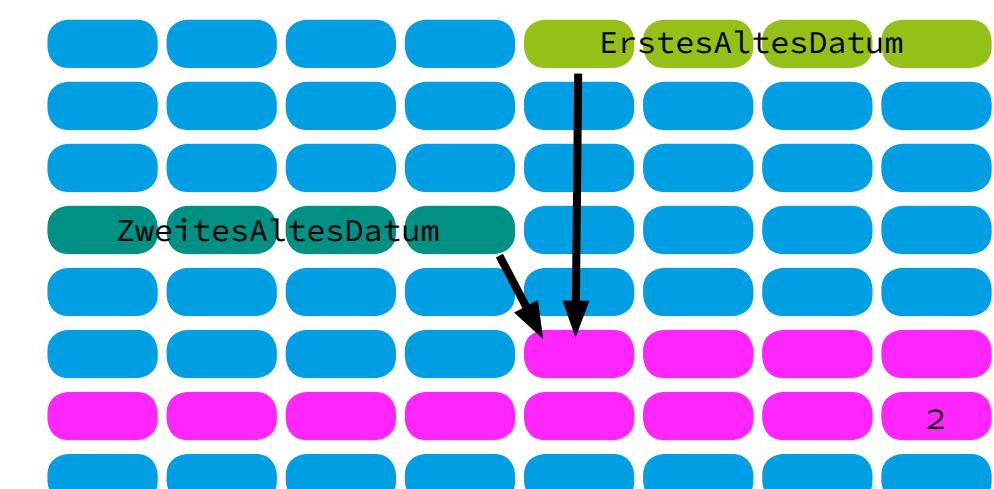
Eine Zeigervariable wurde angelegt, zeigt aber noch nirgends hin. ErstesAltesDatum = Nil!

```
ErstesAltesDatum = New Global.Date
```



Der Speicherbereich für ein neues Objekt wurde reserviert, das Objekt initialisiert und die Zeigervariable zeigt jetzt auf das Objekt. Der Referenzzähler des Objekts wird auf 1 gesetzt.

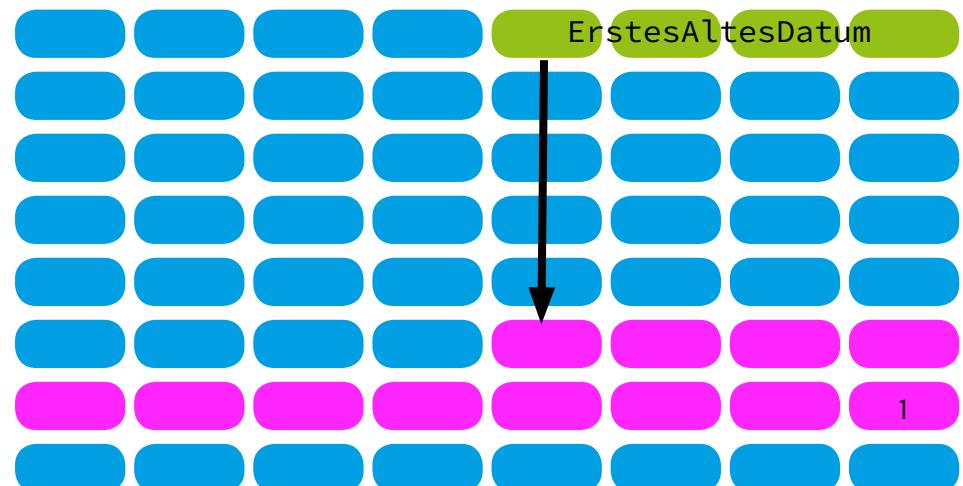
```
Dim ZweitesAltesDatum As Global.Date = ErstesAltesDatum
```



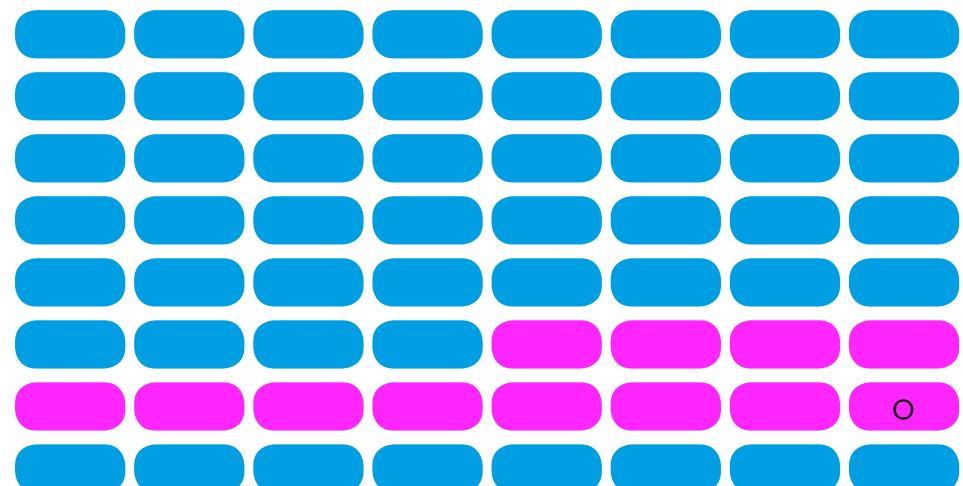
Speicher für die 2. Zeigervariable wurde reserviert, die Zeigervariable auf das Objekt gesetzt und der Referenzzähler um 1 erhöht.



Gerät ZweitesAltesDatum später aus dem Scope, wird der von der Zeigervariablen belegte Speicher freigegeben und der Referenzzähler des Objekts um 1 erniedrigt.



Und wenn auch die erste Referenz nicht mehr benötigt wird, gibt es kurzzeitig diesen Zustand:



Es verweist keine Referenz mehr auf das Objekt, sein Referenzzähler ist bei o. Kurz darauf wird auch der Objektspeicher freigegeben. Wie im Haifischbecken des Turbokapitalismus: Wer keine Referenzen vorzuweisen hat, geht unter ...

3.10.4. Cave leakem!

Normalerweise müssen Sie sich um die Speicherverwaltung also keine großen Gedanken machen. Das erledigt Xojo für Sie, und macht das auch ganz zuverlässig – mit ein paar Ausnahmen, die teilweise allerdings auch den Betriebssystemen anzulasten sind: Diese geben mitunter selbst nicht allen Speicher frei, es kommt zu **Speicherlecks** oder auf Englisch

Memory leaks

Wurde irgendwo ein Fehler gemacht – bei den Xojo-Ingenieuren oder den Betriebssystemkonstrukteuren –, dann kann es sein, dass die Referenzzählung eines Objekts – oder einer Objekt-Property – niemals 0 erreicht. Damit bleibt dann der Speicherbereich reserviert. Werden häufiger Objekte dieser Art erzeugt und nicht wieder freigegeben, klar, dann sammeln sich die Speicherleichen mit der Zeit und sorgen irgendwann dafür, dass entweder Ihr Programm abstürzt (bevorzugt unter 32 Bit, weil dort maximal theoretisch 4 GB pro Programm adressierbar sind – in der Praxis weniger) oder aber andauernd Arbeitsspeicher auf die Festplatte ausgelagert werden muss, um genügend davon für das laufende System frei zu halten. Was auch mit SSDs irgendwann keinen Spaß mehr macht.

In der Regel hilft dann nur ein Beenden des Programms, was dann normalerweise seinen restlichen Speicher freigibt. Manchmal, wenn der Swap-Festplattenarbeitsauslagerungsspeicher zwischenzeitlich angewachsen ist, muss aber auch der Rechner neu gebootet werden. Was es nun also sehr wünschenswert macht, auf Speicherlecks zu achten und diese tunlichst zu vermeiden.

Im Falle von System- oder Xojo-Speicherlecks können Sie in der Regel nicht viel machen, außer natürlich in letzterem Fall einen

Fehlerbericht zu verfassen, damit das Leck von Herstellerseite abgedichtet wird.

Sie selbst können aber jede Menge Prophylaxe betreiben: indem Sie nämlich darauf achten, keine Kreuzverweise zu produzieren.

Das geht leichter, als man denkt:

- ▶ Legen Sie im Projekt mal eine Klasse Elterkasse an, mit einer Property Kind des Typs KindKasse.
- ▶ Das gleiche andersrum: Fügen Sie noch eine Klasse KindKasse ein, die eine Property Elter des Typs Elterkasse besitzt.
- ▶ Dann ergänzen oder ersetzen Sie den Run-Eventhandler mit folgender Schleife:

```
For q As Integer = 0 To 100000000
    Dim e As New Elterkasse
    Dim k As New KindKasse
    e.Kind = k
    k.Elter = e
next
```

- ▶ Klicken Sie auf Run, und es dürfte gar nicht lange dauern, bis Ihnen das Terminal die folgende Fehlermeldung um die Ohren haut:

```
Gleich ist nicht gleich.
debug(78949,0xa356e000) malloc:
*** mach_vm_map(size=1048576) failed (error
code=3)
*** error: can't allocate region
*** set a breakpoint in malloc_error_break to
debug
Segmentation fault: 11
```



In der Schleife wurde dem neuen Elter-Objekt das Kind zugewiesen, und dem Kind-Objekt wiederum das Elter-Objekt. Der Referenzzähler beider Objekte erreicht somit niemals 0, weil Ihre Properties sich gegenseitig referenzieren. Das geht nicht lange gut – wenn beim 32 Bit-Programm die 4 GB-Speichergrenze überschritten wird, mäkelt das Terminal, dass es keinen verfügbaren Speicher mehr gefunden hat.

Fazit: Tun Sie sowas nicht. Referenzieren Sie auf diese Art stets nur in eine Richtung, niemals gegenseitig. Sollte eine Querreferenz einmal nötig sein: Dafür gibt es Abhilfe, die schwachen Referenzen, zu gut Englisch **WeakRef**. Die schauen wir uns zu gegebener Zeit genauer an.

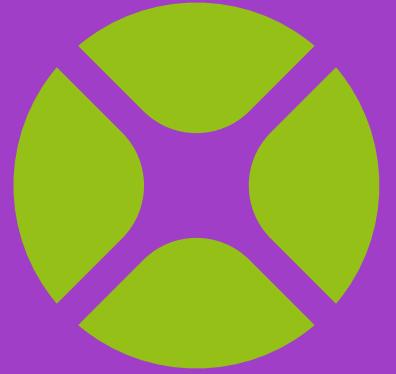


3.11. Ende des Anfangs

Klarer Fall: Es gibt noch vielviel, was sag ich, vielvielviel mehr zu den Objekten, zig weitere Datentypen, und auch aus anderen Gesichtspunkten wäre es vermutlich noch ein bisschen früh, jetzt schon das eigene Softwareunternehmen zu gründen (also sofern Sie den Kapiteln bislang ohne allzu viele weitere Programmierkenntnisse gefolgt sind freilich).

Der reinen Theorie und Vorbereitung war's nun aber wirklich mal genug, finden Sie nicht auch? Jetzt soll es endlich mal in die Praxis gehen, und wie die bei Ihnen bevorzugt aussehen soll, das will ich mir festzulegen nicht anmaßen. Ergo: Freiflug!

Von hier ab sind die Kapitel größtenteils unabhängig zu konsumieren. Querverweise und Abzweigungen werden – hoffe ich – gut ausgeschildert sein. Die Route aber legen ab jetzt Sie fest. Schauen Sie doch vielleicht zunächst in die Details der Sie am stärksten interessierenden Plattform. Oder erweitern Sie in den Referenzkapiteln Ihr Xojo-Vokabular. Vielleicht interessiert Sie auch ein bestimmtes Thema besonders? Das Inhaltsverzeichnis zeigt Ihnen, was schon verfügbar ist.



REFERENZ

Der Nachschlageteil (wird nach und nach gefüllt):

- **Datentypen, Klassen und Befehle im Detail**
- **Index**



Sprachreferenz

Die folgenden Seiten dienen dem schnellen Überblick. Die Sortierung der Sprachelemente folgt dabei den Kategorien der [offiziellen Sprachreferenz](#). Hier zum einfacheren Auffinden als Klickliste:

Befehle

Datentypen



Befehle

Befehle sind all die reservierten Worte, die keine Klassen oder Datentypen darstellen und den Programmablauf beeinflussen – sei es durch Bereitstellung von Variablen, durch deren Berechnung, Schleifenstrukturen etc.

Reservierte Worte können nicht als eigene Variablen verwendet werden – daher sind sie reserviert, nämlich von der Programmiersprache. Der Xojo-Codeeditor hebt reservierte Worte farbig hervor.

Verwandt mit den Befehlen, aber in einer eigenen Kategorie beheimatet, sind die Compiler-Direktiven (aka Pragma-Kommandos).



AddHandler

reserviertes Wort/Befehl

Lenkt die Ausführung eines Events auf einen **Delegate** um, die Adresse einer Methode, die mit **AddressOf** erzeugt wurde.

AddHandler (Objekt.)EventName, AddressOf MethodenName

Soll ein Event, dessen Ausführung mittels AddHandler bereits verbogen wurde, auf eine andere Methode umgelenkt werden, muss die frühere Zuweisung zunächst mittels **RemoveHandler** aufgehoben werden.

Siehe auch „3.8. **Klasse sucht Anschluss**“ auf Seite 87 ff.

FRAMEWORK: N/A

PROJEKTARTEN: ALLE

PLATTFORMEN: ALLE



AddressOf

reserviertes Wort/Befehl

AddressOf erzeugt einen **Delegate** aus einer Methode. Delegates werden im Zusammenhang mit **AddHandler** zum Verbiegen eines Events auf eine Methode oder für Declares und fortgeschritten, modulare Programmiertechniken benutzt.

Die Syntax erschließt sich im Zusammenhang mit Methoden, die einen Delegate entgegennehmen, z. B. beim Nutzen eines programmatisch definierten (nicht im Layout platzierten) Timers:

```
MyTimer = New Xojo.Core.Timer
MyTimer.Period = 1000
MyTimer.Mode = Xojo.Core.Timer.Modes.Multiple
AddHandler MyTimer.Action, AddressOf TimerAction
```

Die Methode **TimerAction** muss als Parameter den Sender des Events entgegennehmen, in diesem Fall also:

```
Sub TimerAction(sender As Xojo.Core.Timer)
    // Irgendeine Aktion
End Sub
```

FRAMEWORK: N/A**PROJEKTARTEN:** ALLE**PLATTFORMEN:** ALLE



As

reserviertes Wort/Befehl

As bestimmt den Datentyp einer Variablen. **As** steht daher niemals allein, sondern ist Teil einer Anweisung, wie

Dim I **As** Integer

FRAMEWORK: N/A

PROJEKTARTEN: ALLE

PLATTFORMEN: ALLE

Siehe auch

Dim



Dim

reserviertes Wort/Befehl

Dim reserviert den Speicherplatz für eine oder mehrere Variablen (des gleichen Datentyps). Es tritt immer zusammen mit **As** auf.

Anwendungen

Dim *VariablenName As Datentyp [= Startwert]*

Dim *VariablenName As [New] Objekttyp*

Dim *VariablenName1, VariablenName2, ... As Datentyp [= Startwert]*

Dim *VariablenName1, VariablenName2, ... As [New] Objekttyp*

VariablenName ist dabei eine frei wählbare Bezeichnung, die mit einem Buchstaben beginnen, keine Leerzeichen und sonst nur weitere Buchstaben, Ziffern und Unterstriche beinhalten darf. Ein Variablenname darf nicht einem reservierten Wort entsprechen.

Datentyp oder **Objekttyp** bestimmen die Art der Variablen oder aller Variablen bei den letzten beiden Anwendungsfällen.

Startwert initialisiert die Variable(n) auf den folgenden Vorgabewert, der dem Datentyp der Variable(n) entsprechen oder implizit in diesen konvertierbar sein muss.

Siehe auch

As

FRAMEWORK: N/A

PROJEKTARTEN: ALLE

PLATTFORMEN: ALLE



Datentypen

Datentypen bestimmen die Art, auf die eine Variable interpretiert werden soll – etwa als Ganzzahl, als Zeichenkette etc.

Das Nichthandbuch übernimmt die Xojo-eigene Unterscheidung von Datentypen und Objekten:
Bei ersteren entspricht der reservierte Speicher direkt dem Variablenwert, bei letzteren ist der eigentliche Datentyp nur ein Ptr auf diesen Speicherbereich.



Array

Sprachkonzept

Ein **Array** (= Gebiet) ist kein Datentyp an sich, sondern eine Sammlung von gleichartigen Datentypen oder Objekten, die über ihren Index (als Integerwert) adressierbar sind.

 **Index** beginnt immer mit 0!

Anwendungen

```
Dim VariablenName([Anzahl]) As Datentyp/Objekttyp
```

VariablenName und **Datentyp/Objekttyp** siehe **Dim**.

Grundsätzlich ist ein Array dynamisch ausgelegt. Es ist also möglich, neue Einträge hinzuzufügen oder bestehende zu entfernen. Mit **Anzahl**, das in runden Klammern stehen muss, kann vorab Speicher in der benötigten Größe für *Anzahl* Datentypen reserviert werden. Dies kann die Ausführungs geschwindigkeit des Programms gegenüber beständigem dynamischen Hinzufügen erhöhen.

Beispiele

```
Dim IntegerArray(2) As Integer
```

reserviert Speicher für zwei Integerwerte unter dem Namen IntegerArray. Diese Werte können unter

IntegerArray(0) und **IntegerArray(1)** gelesen und geschrieben werden, also

```
Dim ErsterWert As Integer = IntegerArray(0)
```

```
IntegerArray(1) = 123
```

Lässt man **Anzahl** weg und gibt nur die Klammern an, oder hat **Anzahl** den Wert **-1**, wird ein Array reserviert, das anfänglich Platz für 0 Werte liefert.

FRAMEWORK:

GLOBAL

PROJEKTARTEN:

ALLE

PLATTFORMEN:

ALLE

Methoden

Append (Wert As Datentyp)

Fügt dem Array einen neuen Wert am Ende hinzu und vergrößert damit das Array um 1. **Wert** muss dem Datentyp des Arrays entsprechen oder transparent in diesen konvertierbar sein:

```
IntegerArray.Append (256)
```

Pop

Das Gegenteil von **Append**: Löscht den letzten Wert des Arrays und verkleinert damit das Array um 1. **Pop** liefert dabei den gelöschten Wert als Rückgabewert der Methode:

```
Dim i As Integer = IntegerArray.Pop
```

Remove (Index As Integer)

Ähnlich wie **Pop**: Entfernt den Wert an Stelle **Index** des Arrays und verkleinert dieses damit um 1. Liefert im Gegensatz zu Pop den gelöschten Wert nicht zurück:

```
IntegerArray.Remove (1)
```

entfernt den 2. Wert des Arrays.



Array

Fortsetzung

Methoden (Fortsetzung)

Insert (Index As Integer, Wert As Datentyp)

Ähnlich wie **Append**, fügt aber den neuen Wert an Position Index ein. **Index** darf nicht größer als **UBound** sein!

```
IntegerArray.Insert (2, 728)
```

fügt den Wert 728 an der dritten Position des Arrays ein und vergrößert damit das Array um 1, wobei sich die nachfolgenden Werte entsprechend verschieben.

UBound As Integer

Liefert den **Index** des letzten Werts des Arrays und damit auch seine Größe:

```
Dim ArrayGröße As Integer = IntegerArray.UBound + 1
```

Sort

Sortiert die Elemente eines Arrays in aufsteigender Reihenfolge:

```
IntegerArray.Sort
```

Funktioniert nur mit Arrays der Datentypen **Text**, **Single**, **Double**, **Integer** (und Subtypen) und **String**.

Benötigen Sie ein absteigend sortiertes Array, können Sie einfach den **Index** vom **UBound** des aufsteigend sortierten Arrays subtrahieren:

```
Dim InversWert As Integer = IntegerArray.Ubound - Index
```

Sort (SortierMethode As Delegate)

Ermöglicht die Sortierung von Arrays nach eigenen Kriterien.

Die **SortierMethode** muss dabei eine Methode sein, die als Parameter zwei Werte des Array-Datentyps entgegennimmt und einen Integerwert zurückliefert. Dieser folgt dabei den folgenden Konventionen:

- 1: Wert1 > Wert2
- 1: Wert1 < Wert2
- 0: Wert1 = Wert2

Beispiel:

Gibt es ein Array DatumArray(100) As Xojo.Core.Date, das 100 verschiedene Daten enthält, kann es mit dem Befehl

```
DatumArray.Sort (AddressOf SortiereDaten)
```

sortiert werden, wenn eine Methode

```
Function SortiereDaten(Wert1 As Xojo.Core.Date, Wert2 As Xojo.Core.Date) _  
As Integer  
    If Wert1.SecondsFrom1970 > Wert2.SecondsFrom1970 Then Return 1  
    If Wert1.SecondsFrom1970 < Wert2.SecondsFrom1970 Then Return -1  
    Return 0  
End Function
```

existiert.



Array

Fortsetzung

Methoden (Fortsetzung)

SortWith (Array1[, Array2, ...])

Führt ein **Sort** des Arrays aus und sortiert die weiteren Arrays **Array1** bis **ArrayN** in der gleichen Reihenfolge.

Beispiel:

```
Dim Nachnamen() As Text = Array ("Mozart", "Beethoven", "Wagner", "Bach")
Dim Vornamen() As Text = Array("Wolfgang Amadeus", "Ludwig van", _
    "Richard", "Johann Sebastian")
Nachnamen.SortWith(Vornamen)
```

Nachnamen ist danach in aufsteigender Reihenfolge sortiert, die Vornamen liegen in korrekt passender Reihenfolge vor:

```
// Nachnamen = {Bach, Beethoven, Mozart, Wagner}
// Vornamen = {Johann Sebastian, Ludwig van, Wolfgang Amadeus, Richard}
```

Array1 bis ArrayN müssen dabei nicht den gleichen Datentyp wie das Haupt-Array besitzen. Bezuglich des Datentyps des Hauptarray gelten die gleichen Einschränkungen wie für **Sort**.

Shuffle

Bringt die Werte eines Arrays in zufällige Reihenfolge:

```
IntegerArray.Shuffle
```

IndexOf (Suchwert As Datentyp[, StartIndex As Integer = 0]) As Integer

Sucht nach dem Vorkommen von **Suchwert** im Array, beginnend bei Index **Startindex** (oder beim ersten Element des Arrays) und liefert dessen Index oder -1, falls nicht gefunden.

```
Dim MeinIndex As Integer = IntegerArray.IndexOf (128)
```

Mehrdimensionale Arrays

Es ist auch möglich, mehrdimensionale Arrays zu definieren:

```
Dim MatrixArray(2, 2) As Integer
```

erzeugt ein Integer-Array mit den Dimensionen (0–1, 0–1).

Allerdings entfällt bei einem mehrdimensionalen Array die dynamische Unterstützung – von den obigen Methoden steht nur **Shuffle** zur Verfügung. Stattdessen muss das Array ggf. mittels **ReDim** neu dimensioniert werden.

 In der Praxis bietet es sich an, ein mehrdimensionales Array eher durch ein Dictionary mit Werten vom Typ **Datentyp()** zu realisieren – die dann eine Dimension des Arrays definieren – oder durch einen MemoryBlock, der den entsprechenden Wert ggf. durch eine zusätzliche Methode herausgibt. Beide Arten sind schneller in der Performance als das mehrdimensionale Array.



Auto

Datentyp

Auto ist ein Platzhalter für jeden beliebigen Daten- oder Objekttyp. Einer Variable des Typs **Auto** kann ein beliebiger Typ zugewiesen werden. Liest man den Wert aus, muss er zunächst einer Variablen des entsprechenden Typs zugeordnet werden, bevor man ihn etwa in einen anderen Datentyp konvertieren kann. Berechnungen beruhend auf dem zugewiesenen Datentyp sind jedoch direkt mit der **Auto**-Variablen möglich:

```
Dim AutoVar As Auto = 42 // womit AutoVar einen Integer-Wert beinhaltet
```

```
Dim ZehnMehr As Integer = AutoVar + 10 // ZehnMehr = 52
```

aber

```
Dim IntWert As Integer = AutoVar
Dim ZahlAlsText As Text = IntWert.ToString
```

oder

```
Dim AutoVar1 As Auto = 5.5 // beinhaltet einen Double-Wert
Dim ZahlAlsText1 As Text = CType(AutoVar1, Double).ToString
```

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**



Boolean

Datentyp

Eine **Boolesche** Variable kann nur zwei Werte annehmen: **True** oder **False**. Ihr Vorgabewert ist **False**.

Intern belegt eine Variable des Types Boolean 1 Byte.

```
Dim IstGespeichert As Boolean = True
```

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE



CFStringRef

Datentyp (fortgeschritten)

CFStringRef ist ein nur unter macOS und iOS verfügbarer Datentyp für Zeichenketten, der im Zusammenhang mit **Declares** für diese beiden Plattformen benutzt wird. CFStringRef konvertiert transparent von und in die Datentypen **Text** und **String**.

```
Public Property Title as Text
    Get
        Declare Function getTitle Lib "AppKit" Selector "title" (Id As Ptr) As CFStringRef
        Return getTitle (ptr(me.Handle))
    End Get

    Set
        Declare Sub setTitle Lib "AppKit" Selector "setTitle:" (Id As Ptr, Value As CFStringRef)
        setTitle (ptr(me.Handle), value)
    End Set
End Property
```

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: MACOS, IOS



CGFloat

Datentyp (fortgeschritten)

CGFloat ist ein nur unter macOS und iOS verfügbarer Datentyp für Fließkommazahlen. Wie **CFStringRef** ist er nur im Zusammenhang mit Declares in diese Betriebssysteme sinnvoll. **CGFloat** ist das Pendant zu **Integer** im Fließkommabereich: Auf einem 32 Bit-System entspricht eine **CGFloat**-Variable einem Single-Wert, unter 64 Bit ist er ein Double.

```
Declare Function initWithComponents Lib "UIKit" Selector_
    "initWithRed:green:blue:alpha:" (obj_id As Ptr, red As CGFloat, green As CGFloat,_
    blue As CGFloat, alpha As CGFloat) As Ptr
```

erzeugt unter iOS ein **UIColor**-Objekt, was der iOS-Klasse zur Verwaltung von Farben entspricht. Ohne **CGFloat** müsste der Declare doppelt definiert werden:

```
#If Target32Bit
    Declare Function initWithComponents Lib "UIKit" Selector_
        "initWithRed:green:blue:alpha:" (obj_id As Ptr, red As Single, green As Single,_
        blue As Single, alpha As Single) As Ptr
#Elseif Target64Bit
    Declare Function initWithComponents Lib "UIKit" Selector_
        "initWithRed:green:blue:alpha:" (obj_id As Ptr, red As Double, green As Double,_
        blue As Double, alpha As Double) As Ptr
#End If
```

 Gleichzeitig mit seiner Einführung wurde **CGFloat** auch schon abgekündigt: Der Datentyp wird in dem Moment entfallen, in dem beide Betriebssysteme ausschließlich auf 64 Bit-Architekturen laufen. Er sollte dann durch Double ersetzt werden.

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: MACOS, IOS



Color

Datentyp

Color ist ein Datentyp zur Speicherung von Farbinformationen (mit optionalem Alpha-Wert für Transparenzen). Die Farbwerte einer **Color**-Variablen lassen sich nicht separat definieren, da **Color** im Gegensatz zu etwa den nativen Farbklassen unter macOS und iOS als Datentyp und nicht als Objekt definiert wurde. Eine Veränderung eines Farbwerts ist nur über die Zuweisungsfunktionen **RGB(A)**, **HSV(A)** und mit Ausnahme von iOS **CMY(A)** oder über die Definition der Farbwerte als Zahl möglich:

```
Dim meineFarbe As Color = &cffff00
```

definiert die Variable **meineFarbe** als Weiß und voll deckend: Das Literal **&c** kündigt einen Farbwert an, der durch die hexadezimalen Werte **&hff** = 255 in Reihenfolge Rot, Grün, Blau und den Wert **&h00** = 0 für die Transparenz bestimmt wird. Entsprechend belegt ein Color-Datentyp 4 Bytes.

Die Angabe des Alpha-Werts ist optional. Wird kein Alpha-Wert definiert, gilt der Vorgabewert 0 = voll deckend.

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**

Methoden

Alpha As Integer

Der Transparenzwert der Farbe als Integer im Bereich 0 (voll deckend) bis 255 (voll transparent)

```
Dim Transparenz As Integer = meineFarbe.Alpha
```

Red, Green, Blue As Integer

Rot-, Grün- und Blauwert einer Farbe als Integer im Bereich 0 (kein Farbauftrag) bis 255 (voller Farbauftrag)

```
Dim Rot As Integer = meineFarbe.Red
```

Cyan, Magenta, Yellow As Double

Cyan-, Magenta- und Gelbwert einer Farbe als Double im Bereich 0 (kein Farbauftrag) bis 1 (voller Farbauftrag)

```
Dim Cyan As Double = meineFarbe.Cyan
```



Color

Fortsetzung

Methoden (Fortsetzung)

Hue, Saturation, Value As Double

Farnton, Sättigung und Helligkeit einer Farbe als Double im Bereich 0 bis 1

```
Dim Hue As Double = meineFarbe.Hue
```

RGB (Red As Integer, Green As Integer, Blue As Integer) As Color

Weist einer Farbe die RGB-Farbdefinitionen im Bereich 0 (keine Farbe) bis 255 (volle Farbe) zu

```
Dim Rot As Color = Color.RGB(255, 0, 0)
```

RGBA (Red As Integer, Green As Integer, Blue As Integer, Alpha As Integer) As Color

Weist einer Farbe die RGB-Farbdefinitionen im Bereich 0 (keine Farbe) bis 255 (volle Farbe) zu, mit zusätzlichem Transparenzwert zwischen 0 (voll deckend) und 255 (voll transparent)

```
Dim Zartblau As Color = Color.RGBA(0, 0, 200, 80)
```

HSV (Hue As Double, Saturation As Double, Value As Double) As Color

Weist einer Farbe die HSV-Farbdefinitionen mit Werten im Bereich 0 bis 1 zu

```
Dim Grüngrau As Color = ColorHSV(0.4, 0.23, 0.5)
```

HSVA (Hue As Double, Saturation As Double, Value As Double, Alpha As Integer) As Color

Wie zuvor, jedoch mit zusätzlichem Transparenzwert im Bereich 0 (voll deckend) bis 255 (voll transparent)

```
Dim ZartGrüngrau As Color = Color.HSVA(0.4, 0.23, 0.5, 100)
```

Unter **Desktop, Kommandozeile und Web**, nicht aber unter iOS, existiert auch die globale Methode

CMY (Cyan As Double, Magenta As Double, Yellow As Double, Alpha As Integer = 0) As Color

Weist einer Farbe die CMY-Farbdefinitionen im Bereich 0 (keine Farbe) bis 1 (volle Farbe) zu, mit optionalem Alpha-Wert für Transparenz

```
Dim Orange As Color = CMY(0, 1, 1)
```

Konstanten

Einige häufig benutzte und betriebssystemabhängige Farben sind als vordefinierte Farbkonstanten verfügbar:

Black, Blue, Brown, Clear, Cyan, DarkGray, Gray, Green, LightGray, Magenta, Orange, Purple, Red, Teal, White, Yellow As Color

```
Dim Grau As Color = Color.Gray
```



CString

Datentyp (fortgeschritten)

CString ist ein Datentyp für Zeichenketten, der in erster Linie für Declares und Manipulationen von MemoryBlocks verwendet wird. Ein **CString** ist **nullterminiert**: Seine Länge wird durch das Auftauchen eines Null-Bytes begrenzt.

```
Dim CS As CString = "Ich bin ein CString"
```

 Wenn Sie sicherstellen wollen, dass wirklich der vollständige Text oder String in einen **CString** konvertiert wird, sollten Sie ihn ggf. vorher auf das Auftauchen von Null-Bytes untersuchen und diese ersetzen. Andernfalls wird der **CString** nur den Originaltext bis zum Auftauchen des ersten Null-Bytes beinhalten.

CString konvertiert, von dieser Einschränkung abgesehen, implizit von und nach **String**.

Zur Konvertierung von und nach **Text** existieren die Methoden **Text.toCString** und **Text.FromCString**.

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**



Currency

Datentyp

Currency ist ein Zahlen-Datentyp speziell zur Berechnung von Geldbeträgen. Intern handelt es sich bei Currency um einen 64-Bit-Integer mit 4 festen Nachkommastellen.

 **Sie sollten immer Currency statt Double oder Single verwenden, wenn Sie mit Geldbeträgen rechnen wollen.** Die „Unschärfe“ durch Rundungsungenauigkeiten bei den Fließkomma-Datentypen werden mit Currency vermieden.

Dim Summe As Currency = 123.45

Currency konvertiert implizit in die anderen Zahlen-Datentypen.

Currency-Datenbereich

Name	Größe	Min.	Max.
Currency	8 Byte	-922.337.203.685.477,5808	+922.337.203.685.477,5807

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE

Methoden

ToText (Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Text
Konvertiert einen Currency-Wert in einen Text, wobei ein optionaler Locale-Parameter die Ausgabe definieren kann:

```
Using Xojo.Core
Dim Betrag As Text = Summe.ToText(Locale.Current)
```

gibt auf einem deutschen System den Text "123,45 €" aus.

gesharete Methoden

FromText (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Currency

Konvertiert einen Text, sofern er einen Zahlenwert repräsentiert, in eine Currency-Variable, wobei die optionale Locale bei der Interpretation der Komma- und Punktbedeutung verwendet wird. Verursacht im Fehlerfall eine Bad Data Exception.

```
Dim Betrag As Text = "123,45"
Dim Rg As Currency = Currency.FromText(Betrag, Xojo.Core.Locale.Current)
```

erzeugt auf einem deutschen System die Variable Rg mit dem Wert 123.4500.



Delegate

Ein **Delegate** ist ein Datentyp, der einen Platzhalter für eine bestimmte Methode darstellt. Delegates bieten die Möglichkeit, Programmcode stark zu flexibilisieren und ihn zugleich übersichtlicher und kürzer zu halten.

Ein Beispiel für die Anwendung eines Delegates liefert die **Sort**-Methode des Array-Datentyps. Der **Delegate** dieser Methode besitzt die Eingabeparameter Wert1 und Wert2 As **Auto** (oder **Variant**) und den Rückgabe-Datentyp **Integer**. Der Sort-Methode kann der **Delegate** jeder eigenen Methode übergeben werden, die dieser Struktur entspricht. Dazu dient der Befehl **AddressOf**, der den **Delegate** der als Parameter übergebenen Sortiermethode erzeugt:

DatumArray.Sort (**AddressOf** SortiereDaten)

Eigene Delegates können in der IDE über einen der Insert-Befehle oder den Add-Button im Code-Editor bzw. im Kontextmenü des Navigators zu einer Klasse oder einem Modul hinzugefügt werden. Der Delegate erscheint dann wie eine Methode im Navigator. Im Inspector können sein Name und eventuelle Ein- und Ausgabeparameter definiert werden, genau wie bei einer normalen Methode.

Um ihn aufzurufen, muss eine weitere Methode mit identischen Parametern deklariert werden. Die Zuweisung erfolgt dann programmatisch durch eine Variable des Delegate-Namens und ihr Aufruf über **Invoke**.

Beispiel:

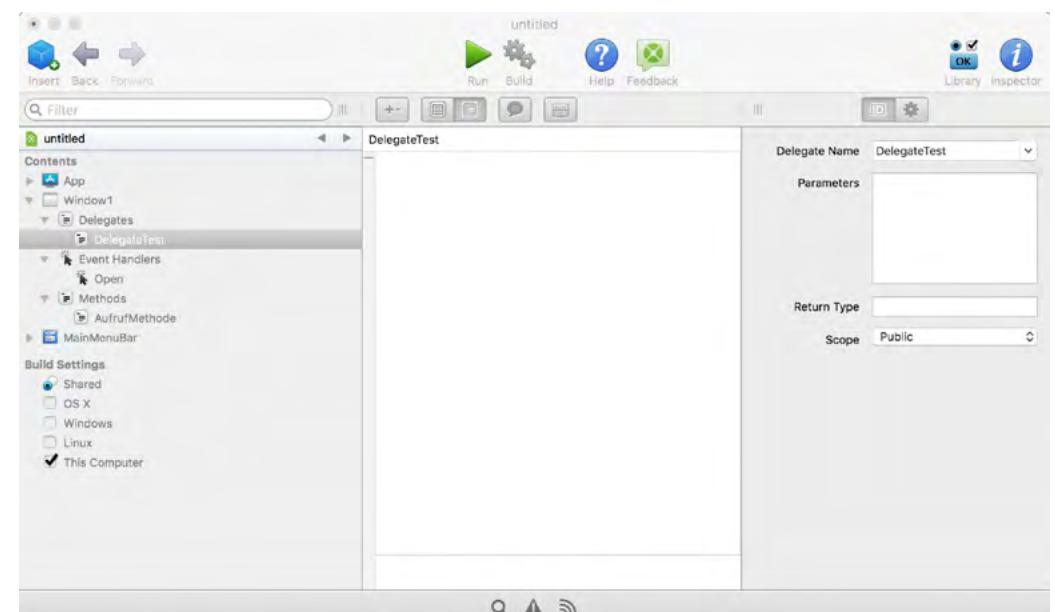
Im Projekt rechts wurde in der IDE der Delegate **DelegateTest** (ohne Parameter) zum Fenster hinzugefügt. **AufrufMethode** bringt eine MessageBox auf den Bildschirm. Der Open-Eventhandler des Fensters beinhaltet

```
Dim Del as DelegateTest = Addressof AufrufMethode
Del.Invoke
```

Auf Run wird die MessageBox der Aufrufmethode angezeigt.

Datentyp (fortgeschritten)

FRAMEWORK:	GLOBAL
PROJEKTARTEN:	ALLE
PLATTFORMEN:	ALLE



Da ein **Delegate** ein **Datentyp** ist, können Sie auch eine **Property** der **Delegate-Unterklasse definieren**. Im obigen Beispiel also z. B. `DelTest As DelegateTest`. Somit kann der Programmcode kontextabhängig eine definierte Funktion aufrufen, ohne dass an sämtlichen Aufrufstellen diese Bedingung überprüft werden müsste. Es wird einfach der von einer zentralen Stelle aus definierte Delegate invoziert.

Methode

Invoke ([optionale Parameter]) [As optionaler Parameter]
Ruft einen Delegate mit seinen Variablen, sofern definiert, auf und übergibt den Rückgabewert, sofern definiert.



Double

Datentyp

Ein **Double** ist ein Datentyp zur Verwaltung von Fließkommazahlen. Er belegt intern 8 Byte. Eine Alternative zum Double stellt **Single** dar. Da die Ausführungsgeschwindigkeiten sich nur marginal unterscheiden, ist die Einsparung eines Doubles durch einen Single nur anzuraten, wenn die höhere Auflösung eines Doubles keinen Vorteil für die Rechnung erbringt und effizientes Speichermanagement benötigt wird.

 **Sie sollten immer Currency statt Double oder Single verwenden, wenn Sie mit Geldbeträgen rechnen wollen.** Die „Unschärfe“probleme“ durch Rundungsungenauigkeiten bei den Fließkomma-Datentypen werden mit Currency vermieden.

```
Dim Pi As Double = 3.14159265358979323846264338327950
```

 **Im Fall von Datenüberschreitungen oder illegalen Operationen können Double-Werte zwei Sonderwerte erhalten, die bei ihrer Konvertierung nach String oder Text offensichtlich werden:**

```
Dim d As Double = -32.24
Dim s As Double = Sqrt(d) // Quadratwurzel einer negativen Zahl = irreale!
Dim t As Text = s.ToString() // t = "NaN" (not a number, keine Zahl)
```

```
Dim d As Double = log(0)
Dim t As Text = d.ToString() // t = "-inf" (negative Unendlichkeit)
```

Double-Datenbereich

Name	Größe	Min.	Max.
Double	8 Byte	$\pm 4,94065645841246544176568792868221372 \cdot 10^{-324}$	$\pm 1,79769313486231570814527423731704357 \cdot 10^{308}$

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE

Methoden

Equals (numValue As Double, maxUIPs As Integer = 1) As Boolean

Vergleicht zwei Double-Werte auf Identität innerhalb einer bestimmten Toleranz. **maxUIPs** bezieht sich dabei auf das letzte Byte des Double-Werts vor dem Exponenten (die [Mantisse](#)). Ein Toleranzwert von 0 liefert das gleiche Ergebnis wie If Double1 = Double2 ...

```
Dim Pi1 as Double = 3.1415926535897932 // letztes Byte hat den Wert &h18
Dim Pi2 As Double = 3.141592653589795 // letztes Byte hat den Wert &h1c
Dim Resultat1 As Boolean = Pi1.Equals (Pi2, 3) // Ergebnis ist False
Dim Resultat2 As Boolean = Pi1.Equals (Pi2, 4) // Ergebnis ist True
```

ToText (Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Text

Konvertiert einen Double in seine Text-Entsprechung. Eine Locale kann angegeben werden, um lokale Eigenarten der Punkt- und Kommasetzung zu berücksichtigen.

```
Dim ZahlText As Text = Pi.ToString()
```



Double

(Fortsetzung)

Methoden (Fortsetzung)

ToText (Locale As Xojo.Core.Locale, Format As Text) As Text

Wie oben, nur muss diesmal eine Locale definiert werden, und der Formatierungstext definiert die Darstellung analog zur [Unicode-Formatierungspalette](#).

```
MsgBox pi.ToText (Xojo.Core.Locale.Current, "#.##") // Ausgabe: 3,14
```

Gesharete Methoden

FromText (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Double

Konvertiert einen Text, **der eine Zahl darstellt**, in seinen Doublewert. Berücksichtigt dabei die optionale Locale zur Interpretation von Komma- und Punktbedeutungen. Entspricht der Text keiner Zahl, ist das Ergebnis 0.

```
Dim Pitext As Text = "3,14159"  
Dim D As Double = Double.FromText(Pitext, Xojo.Core.Locale.Current)
```

D hat auf einem deutschen System den Wert 3.14159.

Parse (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Double

Konvertiert einen Text, **der mit einer Zahl beginnt**, in seinen Doublewert. Berücksichtigt dabei die optionale Locale zur Interpretation von Komma- und Punktbedeutungen. Entspricht der Text keiner Zahl, ist das Ergebnis 0.

```
Dim Geschwindigkeit As Text = "148,5 km/h"  
Dim D As Double = Double.Parse(Geschwindigkeit, Xojo.Core.Locale.Current)
```

D hat auf einem deutschen System den Wert 148.5.



Integer

Datentyp

Ein **Integer** ist ein Datentyp zur Verwaltung von ganzen Zahlen. Von Integer existieren diverse Ableger, die unterschiedlich viele Byte belegen und mit Vorzeichen oder nur im positiven Bereich verwendet werden können – letztere erkennbar durch das vorangestellte „U“ für **unsigned** = **unsigniert**.

Wird der normale Datentyp **Integer** verwendet, ist seine Größe abhängig von der Betriebssystem-Architektur. Auf einem 32 Bit-System belegt ein Integer 4 und auf einem 64 Bit-System 8 Byte, es wird also automatisch ein **Int32** bzw. ein **Int64** daraus.

```
Dim I As Integer = 27
```

Integer-Datenbereiche

Name	Größe	Min.	Max.
Int8	1 Byte	-128	+127
UInt8	1 Byte	0	+255
Int16	2 Byte	-32.768	+32.767
UInt16	2 Byte	0	+65.535
Int32	4 Byte	-2.147.483.648	+2.147.483.647
UInt32	4 Byte	0	+4.294.967.295
Int64	8 Byte	-9.223.372.036.854.775.808	-9.223.372.036.854.775.808
UInt64	8 Byte	0	+18.446.744.073.709.551.615

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE

Methoden

ToBinary (minimumDigits as Integer = 0) As Text

Konvertiert einen Integerwert in seine binäre Textentsprechung, wobei ggf. linksseitige 0en ergänzt werden, um die optionale minimale Anzahl von Stellen zu erreichen.

```
Dim t As Text = I.ToBinary(8) // Ergebnis "00011011" für I = 27
```

ToHex (minimumDigits as Integer = 0) As Text

Konvertiert einen Integerwert in seine hexadezimale Textentsprechung, wobei ggf. linksseitige 0en ergänzt werden, um die optionale minimale Anzahl von Stellen zu erreichen.

```
Dim t As Text = I.ToHex(8) // Ergebnis "0000001B" für I = 27
```

ToOctal (minimumDigits as Integer = 0) As Text

Konvertiert einen Integerwert in seine oktale Textentsprechung, wobei ggf. linksseitige 0en ergänzt werden, um die optionale minimale Anzahl von Stellen zu erreichen.

```
Dim t As Text = I.ToOctal(8) // Ergebnis "00000033" für I = 27
```



Integer

(Fortsetzung)

Methoden (Fortsetzung)

ToText (Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Text

Konvertiert einen Integer in seine Text-Entsprechung. Eine Locale kann angegeben werden.

```
Dim ZahlText As Text = I.ToText // "27" für I = 27
```

ToText (Locale As Xojo.Core.Locale, Format As Text) As Text

Wie oben, nur muss diesmal eine Locale definiert werden, und der Formatierungstext definiert die Darstellung analog zur [Unicode-Formatierungspalette](#).

```
MsgBox I.ToText (Xojo.Core.Locale.Current, "000") // Ausgabe: "027"
```

Gesharete Methoden

FromBinary (theText As Text) As Integer

Konvertiert einen Text, der eine Binärzahl darstellt, in seine Integer-Entsprechung. Erzeugt eine Bad Data Exception, falls der Text keine Binärzahl darstellt.

```
Dim t As Text = "00110101"  
Dim i As Integer = Integer.FromBinary(t) // i = 53
```

FromHex (theText As Text) As Integer

Konvertiert einen Text, der eine Hexadezimalzahl darstellt, in seine Integer-Entsprechung. Erzeugt eine Bad Data Exception, falls der Text keine Hexadezimalzahl darstellt.

```
Dim t As Text = "ff"  
Dim i As Integer = Integer.FromHex(t) // i = 255
```

FromOctal (theText As Text) As Integer

Konvertiert einen Text, der eine Oktalzahl darstellt, in seine Integer-Entsprechung. Erzeugt eine Bad Data Exception, falls der Text keine Oktalzahl darstellt.

```
Dim t As Text = "755"  
Dim i As Integer = Integer.FromOctal(t) // i = 493
```

Parse (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Integer

Konvertiert einen Text, **der mit einer Zahl beginnt**, in seinen Integerwert. Berücksichtigt dabei die optionale Locale. Entspricht der Text keiner Zahl, ist das Ergebnis 0.

```
Dim Bestellung As Text = "22 iMac"  
Dim i As Integer = Integer.Parse(Bestellung) // i = 22
```



OSType

Datentyp (fortgeschritten)

OSType ist ein Datentyp, der einen 4-Buchstaben-Zeichencode in einem Integer-Wert verwaltet. OSType wird von einigen macOS- und iOS-Betriebssystemfunktionen erwartet und hat außerhalb dieser recht wenig Nutzen. OSType konvertiert implizit zu und nach **String**. Ein leerer String ergibt den OSType "????".

OSType belegt 4 Byte.

```
Dim O As OSType = "ABCD"
```

```
Dim S As String = O
```

 Unter **iOS** ist OSType zurzeit nicht nutzbar, da iOS den Datentyp **String** nicht kennt.

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**



Ptr

Datentyp (fortgeschritten)

Ein **Ptr** (gesprochen Pointer) ist ein Datentyp, der auf einen Arbeitsspeicherbereich zeigt. Technisch gesehen ist ein Ptr nichts anderes als ein Integer, und wie dieser belegt er auf einem 32 Bit-System 4 und auf einem 64 Bit-System 8 Byte.

```
Dim p As Ptr = iOSView1.Handle
```

Wie bei Integer lassen sich auch zwischen zwei Ptr **Vergleiche** anstellen (>, <, = etc.), und ihre Werte lassen sich wie bei Integer mathematisch modifizieren (**addieren**, **subtrahieren** etc.).

Ein undefinierter Ptr hat den Vorgabewert **Nil**.

 Historisch bedingt sind die **Handles** der Objekte auf den **Desktop**-Plattformen als **Integer** angelegt, obgleich ihre Funktion freilich das Zeigen auf den Speicherbereich der Objekte ist. So benutzt **Microsoft** in seiner Dokumentation ebenfalls **Integer**-Werte, wenn es um das Deuten auf Speicherbereiche geht, während **Apple** dafür den Datentyp **Ptr** verwendet. Xojo **iOS** verwendet dagegen den Typ **Ptr**.

Für explizite Konvertierungen von Integer nach Ptr kann der Befehl **Ptr()** verwendet werden:

```
Dim i As Integer = 6545265
Dim p As Ptr = Ptr(i)
```

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**

Methoden

Ein Ptr kann in eine Vielzahl von Datentypen konvertiert werden. Alle folgenden Methoden benutzen einen optionalen **Offset As Integer = 0**, der angibt, wie viele Bytes vom Anfang des adressierten Speicherbereichs entfernt der gewünschte Wert extrahiert werden soll. *Exemplarisch:*

```
Dim B As Boolean = P.Boolean(0)
```

Die Rückgabe-Datentypen der Methoden entsprechen ihren Namen:

Boolean(), **CFStringRef()**, **Class()**, **Color()**, **CString()**, **Currency()**, **Double()**, **Int8()**, **Int16()**, **Int32()**, **Int64()**, **Integer()**, **Object()**, **Ptr()**, **Single()**, **String()**, **Text()**, **UInt8()**, **UInt16()**, **UInt32()**, **UInt64()**, **UInteger()**, **Variant()**, **WString()**

 Diese Liste erweitert sich um **Structures**, die Sie in Ihrem Projekt anlegen. Sie können einen Ptr also z.B. durch

```
Dim N As NSRect = p.NSRect(0)
```

in eine Struktur NSRect konvertieren, so diese in Ihrem Projekt definiert wurde.



Single

Datentyp

Ein **Single** ist ein Datentyp zur Verwaltung von Fließkommazahlen. Er belegt intern 4 Byte. Eine Alternative zum Single stellt **Double** dar. Da die Ausführungsgeschwindigkeiten sich nur marginal unterscheiden, ist die Einsparung eines Doubles durch einen Single nur anzuraten, wenn die höhere Auflösung eines Doubles keinen Vorteil für die Rechnung erbringt und effizientes Speichermanagement benötigt wird.

Sie sollten immer Currency statt Double oder Single verwenden, wenn Sie mit Geldbeträgen rechnen wollen. Die „Unschärfe“ durch Rundungsungenauigkeiten bei den Fließkomma-Datentypen werden mit Currency vermieden.

```
Dim Pi As Single = 3.14159
```

Im Fall von Datenüberschreitungen oder illegalen Operationen können Single-Werte zwei Sonderwerte erhalten, die bei ihrer Konvertierung nach String oder Text offensichtlich werden:

```
Dim d As Single = -32.24
Dim s As Single = Sqrt(d) // Quadratwurzel einer negativen Zahl = irreale!
Dim t As Text = s.ToString() // t = "NaN" (not a number, keine Zahl)
```

```
Dim d As Single = log(0)
Dim t As Text = d.ToString() // t = "-inf" (negative Unendlichkeit)
```

Single-Datenbereich

Name	Größe	Min.	Max.
Single	4 Byte	$\pm 1,40129846432481707092372958328991613 \cdot 10^{-45}$	$\pm 3,40282346638528859811704183484516925 \cdot 10^{38}$

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE

Methoden

Equals (numValue As Single, maxUIPs As Integer = 1) As Boolean

Vergleicht zwei Single-Werte auf Identität innerhalb einer bestimmten Toleranz. **maxUIPs** bezieht sich dabei auf das letzte Byte des Double-Werts vor dem Exponenten (die [Mantisse](#)). Ein Toleranzwert von 0 liefert das gleiche Ergebnis wie If Single1 = Single2 ...

```
Dim Pi1 as Single = 3.14159
Dim Pi2 As Single = 3.141589 // letzte Bytes variieren um 5
Dim Resultat1 As Boolean = Pi1.Equals (Pi2, 4) // Ergebnis ist False
Dim Resultat2 As Boolean = Pi1.Equals (Pi2, 5) // Ergebnis ist True
```

ToText (Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Text

Konvertiert einen Single in seine Text-Entsprechung. Eine Locale kann angegeben werden, um lokale Eigenarten der Punkt- und Kommasetzung zu berücksichtigen.

```
Dim ZahlText As Text = Pi.ToString()
```



Single

(Fortsetzung)

Methoden (Fortsetzung)

ToText (Locale As Xojo.Core.Locale, Format As Text) As Text

Wie oben, nur muss diesmal eine Locale definiert werden, und der Formatierungstext definiert die Darstellung analog zur [Unicode-Formatierungspalette](#).

```
MsgBox pi.ToText (Xojo.Core.Locale.Current, "#.##") // Ausgabe: 3,14
```

Gesharete Methoden

FromText (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Single

Konvertiert einen Text, **der eine Zahl darstellt**, in seinen Doublewert. Berücksichtigt dabei die optionale Locale zur Interpretation von Komma- und Punktbedeutungen. Entspricht der Text keiner Zahl, ist das Ergebnis 0.

```
Dim Pitext As Text = "3,14159"  
Dim D As Single = Single.FromText(Pitext, Xojo.Core.Locale.Current)
```

D hat auf einem deutschen System den Wert 3.14159.

Parse (theText As Text, Locale As Xojo.Core.Locale = Xojo.Core.Locale.Raw) As Single

Konvertiert einen Text, **der mit einer Zahl beginnt**, in seinen Doublewert. Berücksichtigt dabei die optionale Locale zur Interpretation von Komma- und Punktbedeutungen. Entspricht der Text keiner Zahl, ist das Ergebnis 0.

```
Dim Geschwindigkeit As Text = "148,5 km/h"  
Dim D As Single = Single.Parse(Geschwindigkeit, Xojo.Core.Locale.Current)
```

D hat auf einem deutschen System den Wert 148.5.



String

Datentyp

Ein String ist ein Datentyp zur Verwaltung von Zeichenketten. Um einen String zu definieren, wird sein Inhalt in geraden doppelten Anführungszeichen angegeben:

```
Dim s As String = "Hallo Welt"
```

Der **Vorgabewert** eines Strings ist ein **leerer String** = "".

 **Zu den Eigenarten eines Strings, insbesondere seiner Codierung, siehe „String vs. Text“ auf Seite 48.** In den meisten Fällen, in denen ein beliebiger Text verwaltet werden soll, bietet sich eher der neuere Datentyp **Text** an.

Es existieren zahlreiche Funktionen zur Stringbe- und -verarbeitung, diese aber als Bestandteil des Frameworks (globale Methoden) und nicht der Klasse selbst zugeordnet. Die Dokumentation des **Text**-Datentyps fasst vieles davon anhand des neueren Datentyps zusammen.

Die eigentlich mathematische **Addition** (+) verknüpft zwei Strings zu einem:

```
Dim s1 As String = "!"  
MsgBox s + s1 // Ausgabe: "Hallo Welt!"
```

Auch lassen sich Strings miteinander via >, <, = etc. vergleichen:

```
Dim A As String = "A"  
Dim B As String = "B"  
If B > A then ... // Ist wahr, da der ASCII-Wert von "B" > Asc("A")
```

Als Größenvergleich werden hier die Wertigkeiten der Zeichen nach ASCII-Reihenfolge genommen. Erwarten Sie also keine lexikalisch korrekte Sortierung nach Duden.

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

ALLE

PLATTFORMEN: ALLE AUSSER IOS

In Xojo erstellte Strings werden intern mit **UTF-8**-Codierung gespeichert. Liest man fremde Strings ein, etwa aus einer Textdatei, muss die **Encoding**-Property korrekt gesetzt werden, um das Auftreten von nicht darstellbaren Zeichen zu vermeiden.

ConvertEncoding bietet die Möglichkeit, die Zeichencodierung eines Strings zu verändern.

Text konvertiert implizit **nach String**:

```
Dim t As Text = "Hallo"  
Dim s As String = t
```

Umgekehrt muss explizit konvertiert werden:

Methode

ToText As Text

Konvertiert einen **String** in eine **Text**-Variable:

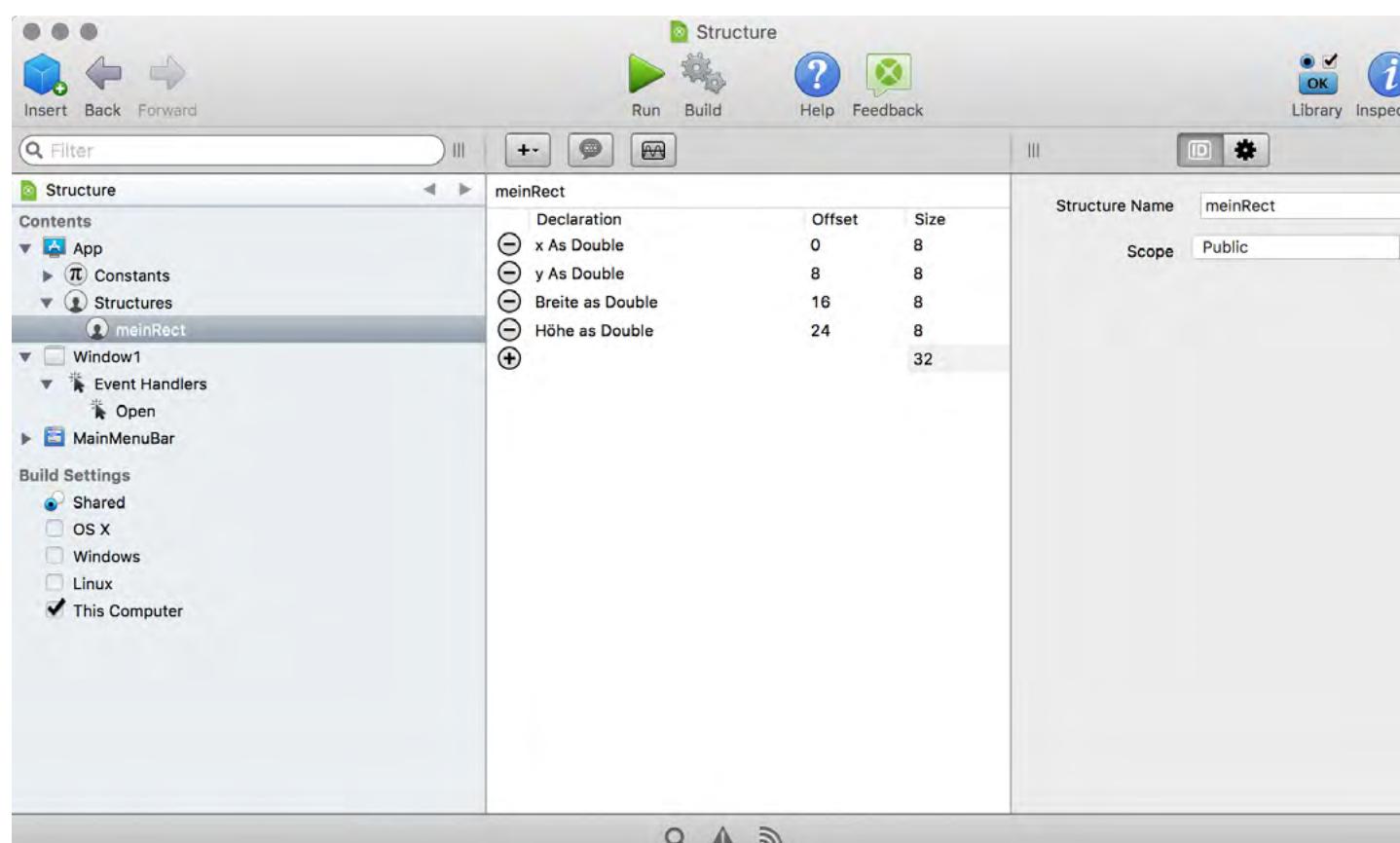
```
Dim s As String = "Hallo"  
Dim t As Text = s.ToText
```



Structure

Eine **Structure** ist ein **zusammengesetzter Datentyp**, der aus einer Aneinanderreihung von anderen Datentypen besteht. Structures können somit alternativ zu **MemoryBlocks** eingesetzt werden, finden Ihren Einsatz aber eher in sehr spezifischen Projekten oder solchen, bei denen API-Funktionen eine Structure erwarten.

Eine Structure wird in der IDE zu einem Modul oder einer Klasse hinzugefügt, indem mit einer der verschiedenen Insert-Arten der Eintrag „Structure“ aufgerufen wird.



```
Dim neuesRect As App.meinRect
neuesRect.x = 23.5 ... // Die anderen Werte können ebenso gesetzt werden.
```

Datentyp (fortgeschritten)

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE

Zeichenketten müssen mit einer definierten Größe zu einer Struktur hinzugefügt werden. Dies geschieht vorzugsweise durch einen Eintrag wie

```
Name(50) As UInt8
```

oder

```
Name As String*50 (nicht möglich bei iOS)
```

für eine Zeichenkette von 50 Zeichen.

```
Dim neuesRect As App.meinRect
neuesRect.x = 23.5 ... // Die anderen Werte können ebenso gesetzt werden.
```

Ein **Ptr, der auf die Adresse einer Struktur deutet** (wie es häufig bei API-Aufrufen der Fall ist), kann analog zu den Ptr-Methoden den entsprechenden Structure-Wert erzeugen, sofern die Structure in seinem Scope liegt – also etwa in einem Modul. Im Beispiel:

```
Dim p As Ptr = ...
// irgendein Funktionsaufruf, der den Ptr zu einer Structure liefert
Dim einRect As App.meinRect = p.meinRect(0)
```



Structure

(Fortsetzung)

Methoden

ByteValue(littleEndian As Boolean) As UInt8()

Gibt die Werte der Structure als Array von UInt8-Werten aus, wobei die [Endianness](#) die Reihenfolge der Bytes vorgibt: Bei **BigEndian** (**littleEndian = False**) wird mit dem höchstwertigen Byte des 1. Wertes begonnen, bei **LittleEndian** = True wird die Byte-Reihenfolge umgedreht (die Werte selbst aber nach wie vor in Struktur-Reihenfolge geliefert)

```
Dim Bytes() As UInt8 = neuesRect.ByteValue(False)
```

ergibt bei nebenstehender Structure ein Array von 32 Bytes: Die 4 x 8 Bytes für die 4 Double-Werte, beginnend mit dem **höchstwertigen Byte von x**. Bei LittleEndian = True würde hier mit dem **niedrigstwertigen Byte von x** begonnen werden.

ByteValue(littleEndian As Boolean, Assigns Value As UInt8())

Die umgekehrte Funktion: Hiermit kann einer Struktur der Inhalt via UInt8-Array zugewiesen werden.

```
neuesRect.ByteValue(False) = Bytes
```

Size As Integer

Liefert die Größe der Structure in Bytes.



Die folgenden beiden Methoden sind **nicht unter iOS verfügbar**:

StringValue(littleEndian As Boolean) As String

Liefert die Werte der Structure als String, unter Berücksichtigung der Endianness.

```
Dim s As String = neuesRect.StringValue(False)
```

StringValue(littleEndian As Boolean, Assigns Value As String)

Weist einer Structure den Inhalt eines Strings unter Brücksichtigung der Endianness zu:

```
neuesRect.StringValue(False) = s
```



Obwohl eine Structure auf den ersten Blick fast wie eine Klasse wirkt, ist sie ein Datentyp. Somit ist es möglich, komplexe Werte als Ergebnis einer Funktion als Structure in einem einzigen Rückgabeparameter zu übergeben. **ByRef** liefert den Ptr auf eine Structure. API-Declares, die Structures verwenden, füllen sie häufig über eine ByRef-Übergabe.



Text

Datentyp

Text ist ein Datentyp zur Verwaltung von Zeichenketten und als solcher der moderne Nachfolger von **String**. Im Gegensatz zu letzterem besitzt eine Text-Variable immer eine bekannte **Zeichencodierung**. Siehe auch „String vs. Text“ auf Seite 48.

```
Dim t As Text = "Hallo Welt"
```

Alle Klassen, Module und Eigenschaften des **Xojo-Frameworks** verstehen sich auf den **Text**-Datentyp, im Gegensatz zum **Classic-Framework**, bei dem **String** zur Zeichenverwaltung verwendet wurde. Der Datentyp Text verwaltet eine Zeichenkette als eine **Reihe von Zeichen** – von **Unicode-Codepoints** (oder auch **Unicode-Skalarwerten**) –, ungeachtet ihres internen Speicherbedarfs. Die Codepoints sind vom Datentyp **UInt32**. **String** verwaltet Zeichenketten als **Reihen von Bytes**, von denen manchmal mehrere zu einem Zeichen zusammengefasst werden müssen, wozu eine Codierung definiert – oder geraten werden muss.

Text konvertiert implizit **nach String**:

```
Dim s As String = t
```

Umgekehrt muss explizit mit der **ToText**-Methode von **String** konvertiert werden.

 **Text** wurde als Bestandteil des **Xojo-Frameworks** eingeführt und folgt dessen Betonung auf **punktnotierter** Schreibung. Entsprechend sind die meisten Befehle zur Textbearbeitung Methoden des Datentyps und keine eigenständigen Sprachbefehle des Frameworks wie bei **String**.

Die eigentlich mathematische **Addition** (+) verknüpft zwei Texte zu einem:

```
Dim t1 As Text = "!"  
MsgBox t + t1 // Ausgabe: "Hallo Welt!"
```

FRAMEWORK: **GLOBAL**
PROJEKTARTEN: **ALLE**
PLATTFORMEN: **ALLE**

Auch lassen sich Texte miteinander via >, <, = etc. vergleichen:

```
Dim A As Text = "A"  
Dim B As Text = "B"  
If B > A then ... // Ist wahr, da der Unicode-CodePoint von "B" > "A"
```

 Ein **Text** ist, wie auch ein **String**, weniger flexibel, als er wirkt – nämlich eigentlich unveränderlich (**immutable**). Fügt man etwa zwei Texte mittels

```
Dim tGesamt As Text = Text1 + Text2
```

zusammen, wird Speicherbereich in Größe von **Text1 + Text2** reserviert und beide Texte dorthin kopiert. Folgen weitere Ergänzungen, wiederholt sich dieser Vorgang komplett, was durch dauerndes Kopieren zu einem vergleichsweise rechenintensiven Vorgang werden kann. Es empfiehlt sich, mehrfach benutzte Texte einmalig als **Konstante** zu deklarieren und ggf. auf effizientere Methoden wie **Join** zurückzugreifen.



Text

(Fortsetzung)

Konstante

CompareCaseSensitive = 1

Wird als Optionsparameter verwendet, um die Unterscheidung von Groß- und Kleinschreibung zu ermöglichen. Dies ist der Wert, der in den Methoden unter **options** eingetragen werden kann – oder 0, um die **standardmäßige Nichtberücksichtigung der Großschreibung** beizubehalten.

Methoden

BeginsWith (other As Text, options As Integer = 0, Locale As Xojo.Core.Locale = Nil) As Boolean

Überprüft, ob der Text mit dem Text **other** beginnt. Wird **CompareCaseSensitive** als **options** übergeben, wird Groß- resp. Kleinschreibung bei der Prüfung berücksichtigt. Die optionale **Locale** erlaubt die Aktivierung sprachspezifischer Unterscheidungskriterien.

```
Dim Original As Text = "Über alle Maßen"
If Original.BeginsWith ("übe") Then ...
```

Ist true, da standardmäßig nicht zwischen Groß- und Kleinbuchstaben unterschieden wird, im Gegensatz zu

```
If Original.BeginsWith ("übe", Text.CompareCaseSensitive) Then ...
// False!
```

Characters As Iterable

Liefert die Zeichen des Textes in ihrer Reihenfolge als iterierbare Texte:

```
Dim t As Text = "Hallo Welt!"
Dim Spiegel As Text
For Each c As Text In t.Characters
    Spiegel = c + Spiegel
Next // Ergebnis: "!tleW ollaH"
```

CodePoints As Iterable

Liefert die Zeichen des Textes in ihrer Reihenfolge als iterierbare UInt32-Werte:

```
For Each codePoint As UInt32 In meinText.Codepoints
    If codePoint = 65 Then MsgBox "Ich habe ein A gefunden!"
Next
```

Compare (other As Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil) As Integer

Führt analog zu **BeginsWith** eine Überprüfung auf Identität durch. Überprüft aber nicht, ob die Textanfänge identisch sind, sondern prüft die komplette Gleichheit der Texte. Liefert als Ergebnis einen Integer-Wert mit folgender Bedeutung:

- 1 Originaltext < other
- 0 Originaltext = other
- 1 Originaltext > other

```
Dim Original As Text = "Über alle Maßen"
If Original.Compare ("übe") < 0 then ... // Ergebnis False, übe < Original!
```



Text

(Fortsetzung)

Methoden (Fortsetzung)

EndsWith (*other As Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil*) As Boolean

Analog zu **BeginsWith**, prüft aber, ob der Text mit **other** endet.

```
Dim Original As Text = "Über alle Maßen"
If Original.EndsWith ("en") Then ... // True!
```

IndexOf (*startPosition As Integer= 0, other As Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil*) As Integer

Findet die Position des ersten Auftauchens eines gesuchten Textes im Originaltext. Liefert **-1**, falls der Suchtext **other** nicht gefunden wurde.

Mit der optionalen Variable **startPosition** kann die Suche erst bei einer bestimmten Zeichenposition gestartet werden. Die anderen Parameter wieder analog zu **BeginsWith**.

```
Dim Original As Text = "Über alle Maßen"
Dim Pos As Integer = Original.IndexOf ("a")      // Pos = 5
Dim Pos1 As Integer = Original.IndexOf (6, "a") // Pos1 = 11
```

Left (*count As Integer*) As Text

Liefert die ersten **count** Zeichen des Originaltextes:

```
Dim Original As Text = "Über alle Maßen"
Dim Über As Text = Original.Left (4)      // Über = "Über"
```

Length As Integer

Liefert die Anzahl der Zeichen im Text:

```
Dim Original As Text = "Über alle Maßen"
Dim Länge As Integer = Original.Length // Länge = 15
```

Lowercase (*locale As Xojo.Core.Locale = Nil*) As Text

Liefert eine Kopie des Textes komplett in Kleinschrift. Die optionale Locale kann wieder zur Berücksichtigung sprachpezifischer Eigenarten hinzugezogen werden:

```
Dim Original As Text = "Über alle Maßen"
Dim Klein As Text = Original.Lowercase // Klein = "über alle maßen"
```

Mid (*start As Integer, optional length as Integer*) As Text

Liefert **length** Zeichen des Textes, beginnend bei Position **start**. Wird **length** nicht definiert, werden alle Zeichen beginnend bei **start** zurückgeliefert:

```
Dim Original As Text = "Über alle Maßen"
Dim Mitte As Text = Original.Mid (5, 4) // Mitte = "alle"
Dim Rest As Text = Original.Mid (5)    // Rest = "alle Maßen"
```



Text

(Fortsetzung)

Methoden (Fortsetzung)

Replace (find as Text, replace As Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil) As Text

Liefert eine Kopie des Textes, bei der **das erste Auftauchen** von **find** durch **replace** ersetzt wurde, respektive den Originaltext, falls **find** nicht gefunden wurde. Die anderen Parameter wieder analog zu **BeginsWith**.

```
Dim Original As Text = "Über alle Maßen"
Dim Ersatz As Text = Original.Replace ("a", "krawa")
// Ersatz = "Über krawalle Maßen" (Was immer das heißt)
```

ReplaceAll (find as Text, replace As Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil) As Text

Liefert eine Kopie des Textes, bei der **jedes Auftauchen** von **find** durch **replace** ersetzt wurde, respektive den Originaltext, falls **find** nicht gefunden wurde. Die anderen Parameter wieder analog zu **BeginsWith**.

```
Dim Original As Text = "Über alle Maßen"
Dim Ersatz As Text = Original.ReplaceAll ("a", "bla")
// Ersatz = "Über blalle Mblaßen" (Ja nun ...)
```

Right (count As Integer) As Text

Liefert die **letzten count** Zeichen des Originaltextes:

```
Dim Original As Text = "Über alle Maßen"
Dim Letztes As Text = Original.Right (5) // Letztes = "Maßen"
```

Split As Text()

Liefert ein **Array of Text**, wobei jedes Element des Arrays ein Zeichen des Originaltextes beinhaltet:

```
Dim Original As Text = "Über alle Maßen"
Dim Split() As Text = Original.Split // Split hat 15 Elemente, beginnend
// mit Split(0) = "Ü"
```

Split (separator as Text, options As Integer = 0, locale As Xojo.Core.Locale = Nil) As Text()

Liefert ein **Array of Text**, wobei die Elemente bei jedem Auftauchen des **separators** getrennt werden. Der **separator** wird nicht zum Teil des Ergebnis-Arrays. **Options** und **locale** analog zu **BeginsWith**.

```
Dim Original As Text = "Über alle Maßen"
Dim Split() As Text = Original.Split (" ") // An Leerzeichen trennen
// Split hat 3 Elemente: "Über", "alle" und "Maßen"
```

TitleCase (locale As Xojo.Core.Locale = Nil) As Text

Liefert eine Kopie des Textes, bei der die Anfangsbuchstaben der Wörter in Großbuchstaben umgewandelt wurden.

```
Dim Original As Text = "über alle maßen"
Dim Titel As Text = Original.TitleCase // "Über Alle Maßen"
```

 Achtung! Hierbei erfolgt **keine Grammatikkontrolle**, sodass der Funktionsname etwas irreführend ist: Auch Verben, Artikel etc. finden sich nach Konvertierung in Großschreibung. Es sei, da man diese amerikanische Schreibung zunehmend sieht, einmal darauf hingewiesen, dass dies im Deutschen **nicht richtig** für Überschriften oder Titel ist!



Text

(Fortsetzung)

Methoden (Fortsetzung)

ToCString (encoding As Xojo.Core.TextEncoding) As CString

Erzeugt einen **CString** aus dem Text unter Berücksichtigung der angegebenen Textcodierung.

```
Dim Original As Text = "Über alle Maßen"  
Dim CS As CString = Original.ToCString(xojo.core.TextEncoding.ASCII)  
// CS = "?ber alle Ma?en" – Die Umlaute sind in reinem ASCII-CString nicht definiert.
```

Trim As Text

Entfernt [Leerzeichen](#) **nach Unicode-Definition** (dazu zählen auch bestimmte Leerzeichen fester Breite, umbruchgeschützte Leerzeichen etc.) von **Anfang und Ende** des Textes:

```
Dim Original As Text = "    Viel    Luft    "  
Dim Trimtext As Text = Original.Trim // Trimtext = "Viel    Luft"
```

TrimLeft As Text

Entfernt Leerzeichen nach Unicode-Definition vom **Anfang** des Textes:

```
Dim Original As Text = "    Viel    Luft    "  
Dim TrimText As Text = Original.TrimLeft // = "Viel    Luft    "
```

TrimRight As Text

Entfernt Leerzeichen nach Unicode-Definition vom **Ende** des Textes:

```
Dim Original As Text = "    Viel    Luft    "  
Dim TrimText As Text = Original.TrimRight // = "    Viel    Luft"
```

Uppercase (locale As Xojo.Core.Locale = Nil) As Text

Liefer eine Kopie des Textes, bei der alle Buchstaben in Großbuchstaben umgewandelt wurden:

```
Dim Original As Text = "Über alle Maßen"  
Dim Groß As Text = Original.Uppercase // = "ÜBER ALLE MASSEN"
```

Gesharete Methoden

FromCString (str as CString, encoding As Xojo.Core.TextEncoding) As Text

Erzeugt einen Text aus dem **CString** unter Berücksichtigung der angegebenen Textcodierung.

```
Dim Original As cstring = "Ich bin ein CString"  
Dim meinText As Text = Text.FromCString(Original, _  
xojo.core.TextEncoding.ASCII)
```

FromUnicodeCodePoint (codepoint as UInt32) As Text

Erzeugt einen Text, der das angegebene Zeichen nach [Unicode-Tabelle](#) beinhaltet:

```
Dim EOL As Text = Text.FromUnicodeCodepoint(10) // EndofLine in Unicode!
```

Join (items() As Text, separator As Text) As Text

Fügt ein Array von Text-Elementen zu einem einzigen Text zusammen, wobei in die „Verbindungsstellen“ der **separator** eingefügt wird.

```
Dim Worte() As Text = Array ("Über", "alle", "Maßen", "erfreut")  
Dim Satz as Text = Text.Join (Worte, " ")
```



WString

Datentyp (fortgeschritten)

Ein **WString** ist ein „Wide Character String“, ein Datentyp zur Verwaltung von Zeichenketten, der hauptsächlich im Rahmen von **Declares für Windows** Anwendung findet. Unter Windows belegt ein WString 2, unter macOS und Linux 4 Bytes pro Zeichen.

```
Dim W As WString = "Hallo Welt!"
```

WString konvertiert implizit nach und von **String** und **Text**:

```
Dim t As Text = "Hallo Welt!"  
Dim w As WString = t  
Dim t1 As Text = w
```

 Unter **Windows** findet ein WString in erster Linie Anwendung, um Variablen des Typs **wchar_t*** zu verwalten. **Byref WString** entspricht der API-Auszeichnung **wchar_t****.

FRAMEWORK:

GLOBAL

PROJEKTARTEN:

ALLE

PLATTFORMEN: ALLE AUSSER IOS



Klassen

Klassen fasst die fortgeschrittenen Datentypen zusammen, die **objektorientiert** aufgebaut sind. Klassen sind dabei die „Baupläne“ zur Erzeugung eines Objekts, das eine Sammlung von Eigenschaften, Methoden und weiteren Code darstellen kann. Im Gegensatz zu einem Modul, das als solches einmalig existiert, können die Objekte von Klassen – die **Instanzen** – in der Regel beliebig oft als eigenständige Objekte existieren. Zugleich bieten Klassen aber auch gemeinsam genutzte Properties und Methoden – Klasseneigenschaften und -Funktionen sozusagen, die für die ganze Klasse gleich sind.

Die objektorientierte Programmierung bietet vielfältige **Vererbungs-** und **Kommunikationsstrukturen** innerhalb der Klassen, die von Xojo reichhaltig unterstützt werden. Grundklasse (**Superklasse**) aller Klassen ist die Klasse **Object**.

Um die Übersicht ein wenig zu erleichtern, werden die in Xojo standardmäßig verfügbaren Klassen kategorisiert aufgelistet. Dabei wird im Großen und Ganzen der Systematisierung der Sprachreferenz gefolgt.

Allen Klassen sind die folgenden Features gemein:

Methoden

Constructor (eventuelle Parameter) As Klasse

Der Constructor ist die Methode, die beim **Instanziieren** eines Klassenobjekts mittels **New** durchlaufen wird. Je nach Klasse sind dabei Initialisierungs-Parameter zu übergeben. In eigenen Klassen muss für diese Methode der Name **Constructor** verwendet werden.

Beim Erstellen eines Constructors einer **Unterklasse** fügt Xojo automatische Kommentare ein, um den Aufruf von **Super.Constructor** zu erleichtern.

```
Dim d As New Date
// benötigt keine Parameter, legt Datum der Gegenwart an

Dim p As New Xojo.Core.Point (50,50)
// Erstellt Point mit den Koordinaten 50; 50
```

Destructor

Der Destructor ist eine optionale Methode, die automatisch durchlaufen wird, wenn ein Objekt aus dem **Scope** gerät – seine Lebenszeit beendet ist und es aus dem Speicher entfernt werden soll. Entsprechend wird ein Destructor nicht via Code aufgerufen, sondern ähnlich wie ein Event vom System gestartet. Hier bietet sich die Möglichkeit, manuell aufzuräumen, Querverkettungen zu lösen etc. Der Destructor muss auf den Namen **Destructor** hören.



Enumeration

Sprachkonzept

Eine **Enumeration** (Nummerierung) ist ein Konzept der objektorientierten Programmierung zur Verwaltung **nummerierter Listen**. Eine Enumeration kann unter anderem verwendet werden, um eine eingeschränkte Auswahl zur Verfügung zu stellen und Fehleingaben auszuschließen – statt eines beliebigen Integerwerts etwa nur Entsprechungen für die Werte 0, 1, 2 und 5.

Eine Enumeration wird **einer Klasse oder einem Modul** in der IDE über eine der **Insert**-Methoden hinzugefügt. Die Enumeration-Werte folgen dabei den Regeln zur Variablenbenennung in Xojo.

Es ist nicht nötig, so wie hier einzelne Integer-Werte zuzuweisen. Standardmäßig beginnt eine Enumeration bei 0 und nummeriert in 1er-Schritten aufsteigend.

 Auch wenn eine Enumeration über Integerwerte definiert wird, ist sie für Xojo ein eigenständiger Datentyp. Es lässt sich jedoch Integer auf einen Enumeration-Wert typecasten, um den Integer-Wert zu erhalten:

```
Dim Wach As Verfassung.Wachheit = Verfassung.Wachheit.mildinteressiert
Dim i As Integer = Integer(Wach) // i = 6
```

Umgekehrt ist dies ebenso möglich:

```
Dim meineWachheit As Verfassung.Wachheit = Verfassung.Wachheit(i)
```

Nicht möglich ist es jedoch, den Begriff als Text zu erhalten. Wird eine solche Funktion benötigt, muss man sich mit einem **Dictionary** oder einer **Select-Case**-Schleife behelfen:

```
Select Case meineWachheit
Case Verfassung.Wachheit.mildinteressiert
    Return "mild interessiert"
End Select
```

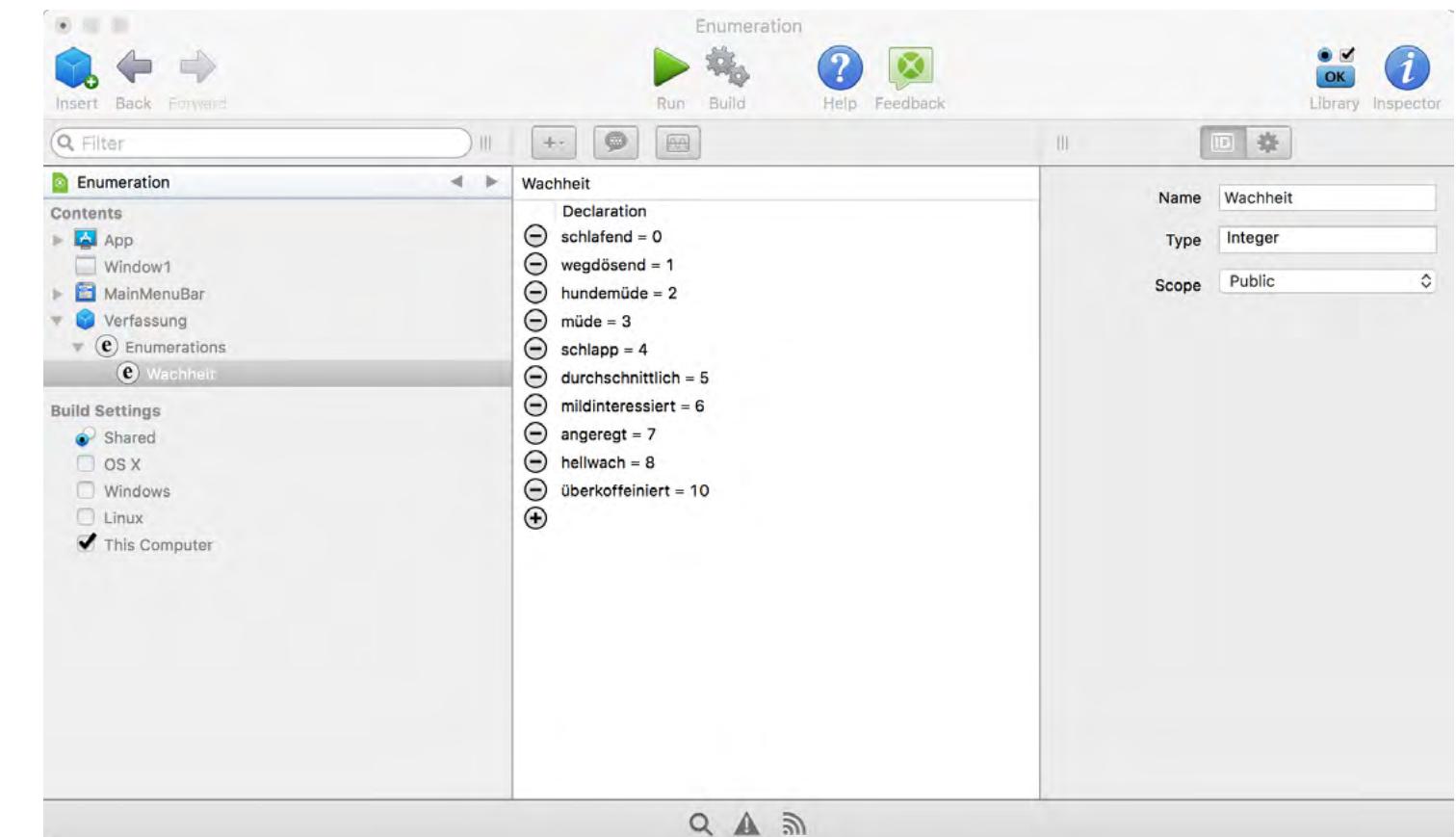


Abb. 13: Enumeration „Wachheit“ als Bestandteil der Klasse „Verfassung“



Klassen (allgemein)

Im Folgenden folgt die Beschreibung der in Xojo verfügbaren Klassen, unterteilt nach Frameworks und Plattformen. Die Grundklasse, auf der alle Klassen beruhen, lässt sich nicht eindeutig zuordnen und wäre in alphabetischer Auflistung nicht gut aufgehoben. Deshalb diese auf der Folgeseite separat.

FRAMEWORK: N/A

PROJEKTARTEN: ALLE

PLATTFORMEN: ALLE



Object

Klasse

Object ist die Superklasse, von der alle anderen Klassen abstammen. Object besitzt keine **Eigenschaften, Methoden und Events**. Als übergeordnete Superklasse aller Klassen eignet sich Object als spezialisierte Alternative zu Variant oder Auto, wenn es um die Verwaltung ausschließlich von Klassen-Objekten im Gegensatz zu auch einfachen Datentypen geht.

Der Vorgabewert eines Objects ist Nil.

```
Dim o As Object = meineKlasse.RückgabeObjekt  
If o <> Nil then ...
```

FRAMEWORK: GLOBAL
PROJEKTARTEN: ALLE
PLATTFORMEN: ALLE



Klassen: Classic-Framework

Das Classic-Framework ist, wie es der Name vermuten lässt, der ältere Bestandteil von Xojo und damit sein RealBasic/Realstudio-Erbe. Es wird nach und nach durch das Xojo-Framework abgelöst, aber noch für lange Zeit weiterexistieren.

Auch wenn dieses Nichthandbuch zum Ziel hat, modernes Xojo zu erklären, kommt man momentan noch nicht am Classic-Framework vorbei, sofern man nicht ausschließlich für iOS programmiert, da die Steuerelemente der Desktop-Plattformen momentan (Stand Xojo 2016r4.1) noch Datentypen wie String verwenden. Es ist angekündigt, dass das Xojo-Framework im Jahr 2017 komplett in die anderen Plattformen integriert werden soll, womit dieser Abschnitt dann demnächst relativ obsolet werden dürfte.

 Unter iOS steht das Classic-Framework nicht zur Verfügung.



Klassen: Classic-Framework: Console

Die Klassen der Kommandozeilen-Plattform bieten auch den Unterbau für Desktop- und Web-Anwendungen. Die Datenbank-Klassen etwa, die der Console zugeordnet werden, stehen natürlich auch für Web- oder Desktop-Anwendungen zur Verfügung. Umgekehrt gilt dies jedoch nicht.



Klassen: Classic-Framework: Desktop

Der größte Unterschied der Desktop-Klassen im Gegensatz zu reinen Kommandozeilen-Möglichkeiten besteht natürlich in der **graphischen Benutzerschnittstelle** und den für diese vorhandenen Steuerelementen.

Grundelement der meisten Desktop-Projekten ist das Fenster – als Klasse **Window** –, wobei dieses nicht zwangsläufiges Charakteristikum einer Desktop-App sein muss – auch reine Menüzeilenprojekte und kleine Widgets wie Dock-Applikationen sind denkbar.

Das Desktop-Framework bietet allgemeine Klassen, Steuerelemente, Dialoge, Globale Methoden, Menüzeilenunterstützung, ein Cursor-Modul und Reportfunktionen.



Klassen: Classic-Framework: Desktop: allgemeine Klassen



Klassen: Classic-Framework: Desktop: Steuerelemente

Steuerelemente – englisch **Controls** – sind die Klassen, die eine Schnittstelle zum Benutzer oder zu Systemdiensten liefern, indem Sie ein visuelles Objekt auf der Benutzeroberfläche darstellen und verwalten.

Ähnlich wie alle Klassen von **Object** abstammen, gibt es auch eine gemeinsame Mutterklasse für **Controls**: **Control**. Die allermeisten Steuerelemente erben von einer weiteren Superklasse, die ihrerseits eine Subklasse von **Control** ist: **RectControl**. Die Properties, Methoden und Events von **RectControl** bei fast allen **Controls** vorhanden – wo Abweichungen existieren, wird dies in den **Control**-Beschreibungen vermerkt. Verfolgen Sie daher immer die Hierarchieangaben, um sämtliche Features zu erfassen.



Control

Klasse/Steuerelement

Control ist die Basisklasse, von der alle Steuerelemente der Desktop-Plattformen erben. Sie bildet die Basis für **RectControl**, die die Grundklasse der meisten Desktop-Steuerelemente bildet. Achten Sie daher auf die Superklasse-Angaben, um sämtliche Features eines Steuerelements zu erfassen!

Ein Control kann nicht direkt erzeugt werden, sondern entsteht durch Instanziierung einer seiner Subklassen.

Control bietet die grundlegenden Events für Steuerelemente: **Open** und **Close**, sowie eine Property **Handle** – den Ptr auf das Objekt selbst, vor allem wichtig für Declares, um weitere Features zu nutzen –, die **Mausposition**, Properties zur Feststellung des Fensters, zu dem das Control gehört, und eine **Close**-Methode, um ein Schließen des Controls zu erzwingen.

Events

Open()

Feuert, wenn das Control initialisiert wird. In diesem Event können z. B. Initialisierungen ausgeführt werden, die über die **Inspector Behavior Settings** nicht möglich sind.

 **Achtung!** Sie sollten hier besser keine Initialisierungen starten, die auf **andere Controls** des gleichen Fensters zurückgreifen. Diese müssen noch nicht zwangsläufig initialisiert sein. Benutzen Sie für solche Initialisierungen besser den **Open**-Event der **Window**-Klasse.

Close()

Das Steuerelement wird geschlossen. Wenn Sie Properties aufräumen, Querverbindungen lösen oder weitere Vorgänge auslösen müssen, die Referenzen zum Control lösen: Hier ist der richtige Platz dafür!

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

DESKTOP

PLATTFORMEN: **ALLE AUSSER IOS**

OBJECT

SUPERKLASSE:

Properties

Handle As Integer

Der Zeiger auf das Control an sich. Siehe auch **Ptr**.

 Liefert unter **macOS** (außer für die Toolbar) den Zeiger auf das **NSView**, unter **Windows** das **HWND** und unter **Linux** das **GtkWidget**.

Index As Integer

Der Index des Controls, falls es in einem **Control Set** eingebunden ist.

MouseX As Integer

Die derzeitige X-Koordinate des Mauszeigers, gemessen von der oberen linken Ecke des Fensters.

MouseY As Integer

Die derzeitige Y-Koordinate des Mauszeigers, gemessen von der oberen linken Ecke des Fensters.



Control

(Fortsetzung)

Properties (Fortsetzung)

Name As String

Der Name des Steuerelements. Kann nur in der IDE im Inspector gesetzt werden.

PanelIndex As Integer

Falls sich das Control auf einem TabPanel oder PagePanel befindet, ist dieser Wert gleichbedeutend mit dem Index des Mutterobjekts (also dem Tab oder der Seite, auf der es sich befindet). Andernfalls ist dieser Wert –1.

Scope As Scope

Der Scope des Controls, also seine Erreichbarkeit von außerhalb (Public, Protected oder Private). Kann nur in der IDE im Inspector gesetzt werden. Siehe „Was deins ist, ist auch meins, aber was meins ist,“ auf Seite 48.

Window As Window

Das Window, zu dem das Control gehört. Auch wenn diese Property nur lesbar ist, können dessen Properties selbst benutzt werden:

```
me.Window.Top = me.Window.Top + 10  
// verschiebt das Mutter-Window 10 pt tiefer
```

Verwechslungsgefahr! Dieses Window muss nicht mit dem höchstrangigen Fenster identisch sein! Es kann sich hier z. B. auch um ein als Sheet in einem weiteren Fenster eingebettet ist. Vgl. TrueWindow der RectControl-Klasse.

Methode

Close

Schließt das Control und entfernt es aus dem Speicher. Feuert den Close-Event vorher.

Ist das Control Teil eines Control Sets, eines TabPanels oder PagePanels, werden dessen Indizes neu nummeriert, um keine Lücken entstehen und die Nummerierung immer bei 0 beginnen zu lassen.



RectControl

Klasse/Steuerelement

RectControl ist eine weitere Basisklasse der meisten Desktop-Steuerelemente. Ein RectControl kann nicht direkt erzeugt werden – es wird implizit durch Instanziierung seiner Subklassen angelegt.

RectControl liefert die Basisfunktionen für die Verwaltung eines rechteckigen Bildschirmschnitts. Dazu gehören Events wie die Registrierung von Mausbewegungen, Klicks und Tastatureingaben innerhalb der eigenen Grenzen, Methoden zum Handling von Drag & Drop-Features und Properties zur Bestimmung der eigenen Dimensionen und der Steuerelemente-Hierarchie.

Events

ConstructContextMenu (Base As MenuItem, x As Integer, y As Integer) As Boolean

Feuert, wenn das Control ein Kontextmenü aufbaut. Je nach Plattform mag der Auslöser ein Rechtsklick sein oder die Betätigung der Kontextmenü-Taste auf der Tastatur.

X und Y sind die **Mausposition**, falls ein Mouse-Event der Auslöser war – andernfalls sind sie -1.

Base ist das grundlegende **MenuItem**, an das Sie an dieser Stelle weitere MenuItems anhängen können.

Bei Übergabe von **False** als Rückgabewert wird der Event in der Steuerelemente-Hierarchie weiter „nach oben“ gereicht. True lässt das **ContextMenu** erscheinen.

```
Base.Append (New MenuItem( "mein Eintrag"))
Return True
```

lässt ein **ContextMenu** mit (mindestens) dem MenuItem „mein Eintrag“ erscheinen – weitere können vom System vorgegeben sein. Auf das Menü reagiert werden kann im Event

FRAMEWORK:

PROJEKTARTEN:

PLATTFORMEN: **ALLE AUSSER IOS**

SUPERKLASSE:

CLASSIC

DESKTOP

CONTROL

ContextualMenuItemAction (HitItem As MenuItem) As Boolean

Feuert, wenn das Kontextmenü-MenuItem HitItem vom Benutzer ausgewählt und dieses nicht von seinem Action-Eventhandler beantwortet wurde. Analog zu ConstructContextMenu bricht der Rückgabewert True die weitere Bearbeitung des Events in der Steuerelemente-Hierarchie ab.

```
If HitItem.Text = "mein Eintrag" Then
    meineMethode
    Return True
End If
```

DragEnter (obj As DragItem, Action As Integer) As Boolean

Feuert, wenn der Benutzer ein DragItem in das Rechteck des RectControls zieht. Der Action-Wert entspricht den Konstanten der **DragItem**-Klasse und definiert, welcher Art die Drag-Action ist (0 = Default, 1 = Copy, 2 = Move, 3 = Link).

Die Rückgabe von True verhindert das Entgegennehmen des DragItems.

```
If Obj.Picture = Nil then Return True
// akzeptiert nur Bilder
```



RectControl

Fortsetzung

Events (Fortsetzung)

DragExit (obj As DragItem, Action As Integer)

Informiert darüber, dass das DragItem obj den Rahmen des RectControls verlassen hat. Action siehe DragEnter.

DragOver (obj As DragItem, Action As Integer) As Boolean

Das DragItem obj wurde innerhalb der Grenzen des RectControls bewegt. Action und Rückgabewert siehe DragEnter.

DropObject (obj As DragItem, Action As Integer)

Das DragItem obj wurde innerhalb der Grenzen des RectControls „fallen gelassen“. Action siehe DragEnter.

 Dieser Event kann nur stattfinden, wenn in den DragEnter- oder DragOver-Events die weitere Bearbeitung nicht durch Rückgabe von True verhindert wurde. Siehe AcceptFileDrop u. ä.

KeyDown (Key As String) As Boolean

Der Benutzer hat die Taste(nkombination) gedrückt, die zum Zeichen Key führt, während das RectControl den Focus hatte. Rückgabe von True verhindert die weitere Bearbeitung des Tastatur-Events in der Steuerelemente-Hierarchie.

KeyUp (Key As String)

Der Benutzer hat die Taste(nkombination) losgelassen, die zum Zeichen Key führt, während das RectControl den Focus hatte. Durch Veränderungen der Sondertasten kann Key ein anderer sein als beim KeyDown-Event.

MouseEnter

Der Mauszeiger hat den Bereich des RectControls betreten.

MouseExit

Der Mauszeiger hat das Gebiet des RectControls verlassen.

MouseMove (X As Integer, Y As Integer)

Der Mauszeiger wurde innerhalb der Grenzen des RectControls bewegt. X und Y sind relativ zum RectControl selbst, nicht die allgemeinen oder fensterbezogene Koordinaten.

MouseWheel (X As Integer, Y As Integer, DeltaX As Integer, DeltaY As Integer) As Boolean

Das Scrollrad der Maus wurde benutzt. X und Y siehe MouseMove. DeltaX und DeltaY beinhalten die Scrollzeilen, um die das Scrollrad in horizontaler und vertikaler Richtung bewegt wurde, so wie vom Betriebssystem und den individuellen Mauseinstellungen im System vorgegeben. Sie folgen dem Koordinatensystem von Xojo:

DeltaX > 0: Bewegung nach rechts

DeltaX < 0: Bewegung nach links

DeltaY > 0: Bewegung nach unten

DeltaY < 0: Bewegung nach oben

Die Rückgabe von True stoppt die weitere Bearbeitung des Events.

 MouseDown ist kein Event der RectControl-Klasse, sondern individuell bei diversen Unterklassen zu finden.



RectControl

Fortsetzung

Properties

Active As Boolean

Gibt an, ob das RectControl aktiv ist. Das ist dann der Fall, wenn sein Fenster im Vordergrund ist. Ein nichtaktives RectControl wird häufig (und je nach Plattform) anders dargestellt als ein aktives, es sei denn, seine AutoDeactivate-Property ist False.

AutoDeactivate As Boolean

Bestimmt, ob das RectControl anders dargestellt werden soll, wenn es sich auf einem Fenster im Hintergrund befindet – nicht auf dem aktiven Vordergrund-Fenster. Vorgabewert True. Siehe Active.

Enabled As Boolean

Bestimmt, ob das RectControl „betriebsbereit“ ist – ist diese Property False, reagiert das RectControl nicht auf Klicks und kann nicht den Focus erhalten. Vgl. Visible.

Left, Top, Height, Width As Int16

Die Dimensionen des RectControls: Seine linke und seine obere Position in Relation zum Fenster sowie Höhe und Breite.

LockBottom, LockTop, LockLeft, LockRight As Boolean

Die Verriegelungsoptionen für die untere, obere, linke und rechte Seite des RectControls in Relation zu seinem Mutterobjekt. Identisch mit den im Inspector graphisch zu setzenden Schloss-Symbolen.

MouseCursor As MouseCursor

Die Form, die der Mauszeiger annehmen soll, wenn er innerhalb der Grenzen des RectControls positioniert ist und die MouseCursor-Properties sowohl des Application-Objekts als auch des Windows Nil sind.

```
me.MouseCursor = System.Cursors.FingerPointer
```

im Open-Event des RectControls lässt – sofern obige Bedingungen erfüllt sind – den Mauszeiger die Form der zeigenden Hand annehmen.

Parent As RectControl

Das Mutterobjekt des RectControls. Hiermit lässt sich die Steuerelemente-Hierarchie auch jenseits der IDE beeinflussen:

Positioniert man im Layout-Editor ein RectControl vollständig innerhalb eines anderen RectControls, so wird das andere RectControl zum Mutterobjekt – das Kind-RectControl wird in ihm eingebettet, was während der Ausrichtung in der IDE durch eine rote Umrandung des Mutterobjekts dargestellt wird. Die Locking-Properties beziehen sich dann auf das Mutterobjekt und sind nicht mehr in Relation zum Fenster zu betrachten.

```
me.Parent = Nil
```

löst das RectControl aus seinem Elternobjekt und lässt es direktes Kind des Fensters werden.



RectControl

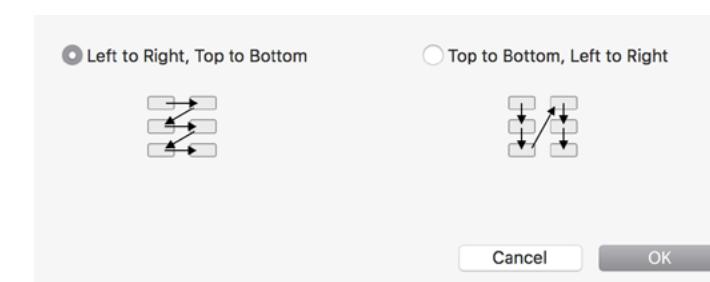
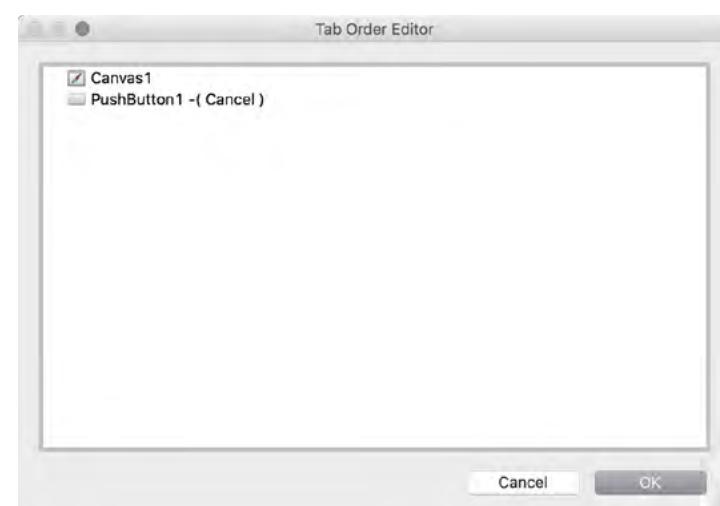
Fortsetzung

Properties (Fortsetzung)

TabIndex As Integer

Der TabIndex regelt die Reihenfolge, in der der Benutzer mittels Druck auf die Tabulator-Taste den Focus auf das nächste RectControl legen kann – bzw. mittels Hochstelltaste + Tab zurück. Siehe TabStop.

 In den meisten Fällen bietet es sich an, eher den TabIndex im Inspector zu setzen, zumal die IDE Funktionen zum automatischen und manuellen Nummerieren des TabIndex bereitstellt:



TabStop As Boolean

Bestimmt, ob das RectControl über den TabIndex erreicht werden kann. Ist diese Property True (der Standard), nimmt sie daran teil. Ist sie False, kann das RectControl nicht über Eingabe von Tab angesprungen werden, aber ggf. programmatisch oder durch Klick den Focus erhalten.

TrueWindow As Window

Liefert das höchststrange Fenster, also die Klasse Window, auf dem das RectControl beheimatet ist, im Gegensatz zur Property Window der Control-Superklasse, die das direkte Mutterfenster erkennen lässt, das seinerseits eingebettet sein kann.

```
me.TrueWindow.Top = me.TrueWindow.Top + 10
// verschiebt das Fenster 10 pt tiefer
```

Visible As Boolean

Falls False, ist das RectControl unsichtbar. Vgl. Enabled.

Methoden

AcceptFileDrop (FileType As String)

Erlaubt Drag & Drop-Aktionen mit Dateien der unter FileType definierten Art(en). FileType muss entweder über die FileType-Klasse oder in der IDE im FileType Sets Editor definiert werden.

```
Dim pngType As New FileType
pngType.Name = "image/png"
pngType.MacType = "PNG"
pngType.MacCreator = "ogle"
pngType.Extensions = "png"
meinControl.AcceptFileDrop ("image/png")
```

ermöglicht das Drag & Drop von PNG-Dateien auf meinControl. Vgl. DropObject u. Drag-Events.



RectControl

Fortsetzung

Methoden (Fortsetzung)

AcceptPictureDrop

Ermöglicht Drag & Drop von Bildobjekten aus anderen Applikationen, wie das Ziehen eines Bildes aus dem Webbrowser direkt auf das RectControl. Sollen (auch) Bilder aus dem Dateiverwaltungsprogramm des Systems (Finder, Windows Explorer, ...) dragbar sein, muss stattdessen (bzw. zusätzlich) AcceptFileDrop benutzt werden.

AcceptRawDataDrop (FileType As String)

Erlaubt Drag & Drop-Aktionen von Daten der unter FileType definierten Art(en). FileType muss entweder über die FileType-Klasse oder in der IDE im FileType Sets Editor definiert werden.

```
meinControl.AcceptRawDataDrop("????")
```

erlaubt das Ziehen von Daten des unter der IDE definierten FileTypes ("????") auf meinControl.

AcceptTextDrop

Erlaubt Drag & Drop-Aktionen von Texten auf das RectControl.

DrawInto (G As Graphics, X As Integer, Y As Integer)

Zeichnet den Inhalt des RectControls an der Position X, Y in das Graphics-Objekt G.

```
Dim p As New Picture(Me.Width, Me.Height)
meinControl.DrawInto(p.Graphics, 0, 0)
Canvas1.Backdrop = p
```

erzeugt ein Bild des Aussehens von meinControl und definiert es als Hintergrundbild von Canvas1.

Invalidate (EraseBackground As Boolean = True)

Markiert das RectControl als dirty. Damit wird das System angewiesen, es beim nächsten Bildaufbau neu zu zeichnen, was bei einem Canvas z. B. den Paint-Event auslöst.

Wird EraseBackground = False übergeben, so wird die Fläche des RectControls nicht vorher gelöscht.

Invalidate (X As Integer, Y As Integer, Width As Integer, Height As Integer, EraseBackground As Boolean = True)

Wie oben, nur wird hier nur ein bestimmtes Rechteck des RectControls als dirty gekennzeichnet.

Refresh (EraseBackground As Boolean = True)

Hiermit wird ein sofortiger Bildneuaufbau des RectControls ausgelöst, also im Gegensatz zu Invalidate nicht auf den Neuaufbau des Systems gewartet. Siehe Invalidate.



Achtung! Obwohl es so klingt, als würde Refresh zu schnelleren Resultaten führen, kann sein Gebrauch zur Verlangsamung des Bildaufbaus führen, insbesondere, wenn es häufig aufgerufen wird. Somit mag z. B. ein Paint-Event eines Canvas mehrfach aufgerufen werden, obwohl das System selbst nur einen einzigen Bildschirm-Neuaufbau in dieser Zeit durchführt. Es empfiehlt sich daher in der Regel, eher Invalidate zu verwenden!

RefreshRect (X As Integer, Y As Integer, Width As Integer, Height As Integer, EraseBackground As Boolean = True)

Wie oben, nur wird hier nur ein bestimmtes Rechteck des RectControls zum sofortigen Neuaufbau bestimmt.



Kleine Inkonsistenz in der Sprachlogik, finde ich. Stringenter sollte das 2. Invalidate auch InvalidateRect heißen oder diese Methode ebenfalls nur Refresh.



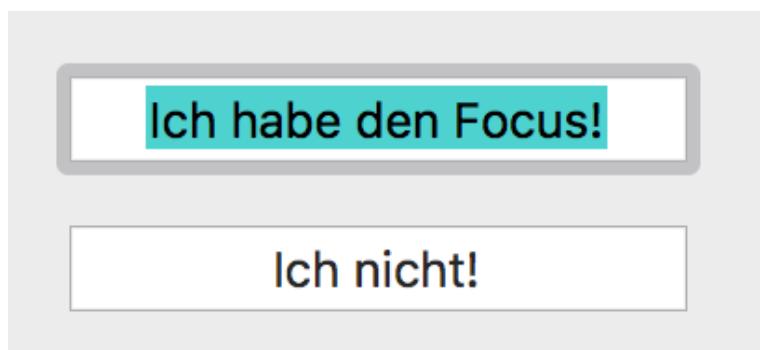
RectControl

Fortsetzung

Methoden (Fortsetzung)

SetFocus

Setzt den Focus auf das RectControl, sofern möglich. Damit werden Tastatureingaben zuerst von diesem RectControl entgegengenommen – siehe KeyDown und KeyUp. Sofern es das RectControl unterstützt, wird außerdem ein FocusRing um es herum gezeichnet – eine Hervorhebung, die es als „ausgewählt“ kennzeichnet. Die Textauswahl wie unten ist kein Kennzeichen des Focus. Sie hat sich durch den Screenshot ergeben.



Enumerationen

Viele RectControl-Unterklassen verwenden die folgende Enumeration:

FontUnits

0 = Default	=	Systemvorgabe.
1 = Pixels	=	Pixel
2 = Points	=	Point
3 = Inches	=	Inch/Zoll
4 = Millimeter	=	mm

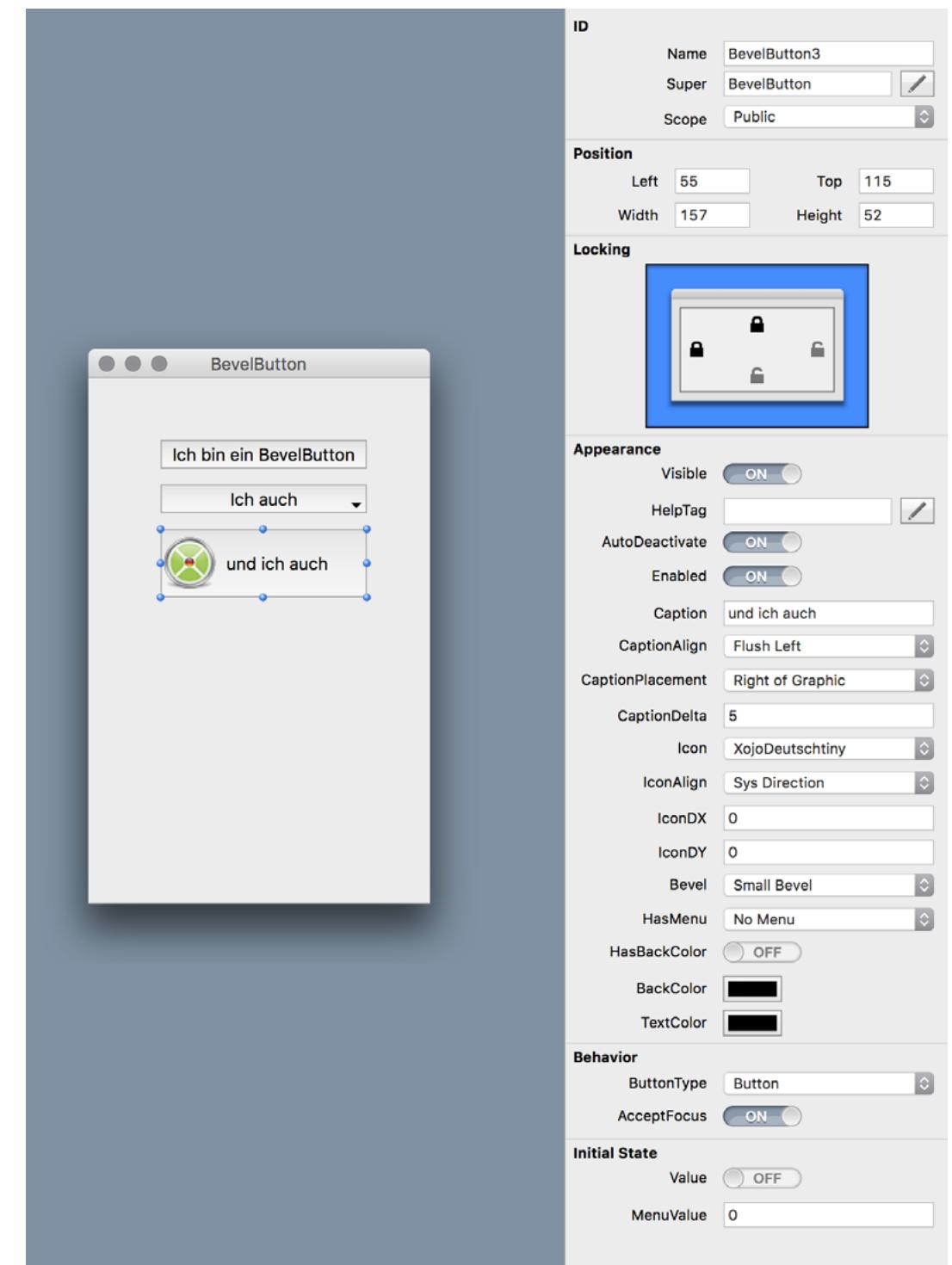


BevelButton

Einen BevelButton (engl. to bevel = abkanten oder fräsen) kann man als Erweiterung des geläufigeren PushButtons betrachten: Ist letzterer ein textbeschrifteter Taster, kann der BevelButton Text, Grafik, ein PopUpMenu oder Kombinationen dieser beinhalten und nicht nur als Taster, sondern auch als Schalter eingesetzt werden.

Seine Erweiterungen gegenüber dem PushButton bestehen daher in einer booleschen Value-Property für den Schaltwert (an oder aus), einer Icon-Property für ein optionales Bild, Properties für das Menü und Hintergrundfarbe sowie Positionierungs-Properties für die Inhalte.

Seine Grundform ist **nicht höhenlimitiert** wie die normale PushButton-Form. Man kann einen BevelButton auf jede gewünschte Größe ziehen.



Klasse/Steuerelement

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

DESKTOP

PLATTFORMEN: ALLE AUSSER IOS

SUPERKLASSE: RECTCONTROL



BevelButton

Fortsetzung

Events

Action

Feuert, wenn der Button mit der Maus geklickt, seine Push-Methode benutzt oder sein Accelerator Key gedrückt wurde.

GotFocus

Der Button hat den Focus bekommen – er wird entsprechend durch einen Rahmen markiert.

Das funktioniert auf dem Mac nur, wenn in der Systemsteuerung/Tastatur/Kurzbefehle „Alle Steuerungen“ aktiviert ist. Siehe PushButton.

LostFocus

Der Button hat den Focus wieder verloren. Siehe GotFocus.

MouseDown (X As Integer, Y As Integer) As Boolean

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum BevelButton selbst) gedrückt. Der Rückgabewert True signalisiert, dass dieser Event selbst bearbeitet wird und keine weitere Verarbeitung durch das System mehr erfolgen soll. Infolgedessen wird der Action-Event nicht feuern. Stattdessen wird der MouseUp-Event feuern, der beim Standard-Rückgabewert False durch den Action-Event sonst nicht zum Zuge käme. Siehe PushButton/MouseDown.

MouseUp (X As Integer, Y As Integer) !

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum BevelButton selbst) losgelassen. Feuert nur, wenn MouseDown True zurückliefert.

Properties

AcceptFocus As Boolean

Bestimmt, ob der BevelButton den Focus erhalten kann. Meistens wird diese Property in der IDE gesetzt.

BackColor As Color !

Definiert die Hintergrundfarbe des Buttons. Um diese anzuzeigen, muss die HasBackColor-Property True sein. Meistens wird diese Property in der IDE gesetzt.

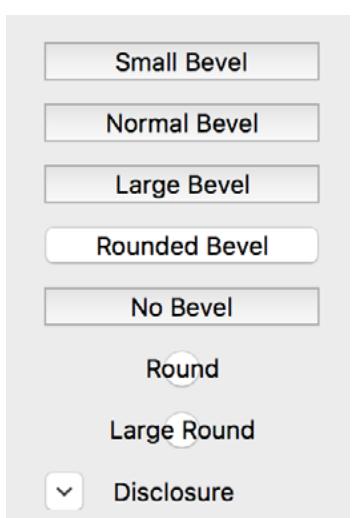
Diese Property existiert nur unter Windows und Linux. Mit macOS funktioniert die Hintergrundfarbe nicht. Sie wird vom Standardrahmen des Buttons überschrieben, auch wenn dieser auf "No Bevel" eingestellt ist. Behelfen Sie sich hier mit einem flächendeckenden, einfarbigen Picture als Icon (das Sie auch programmatisch anlegen können).

Bevel As Integer !

Der Rahmen- (bzw. wörtlich eher Fräungs-)stil des BevelButtons. Hiermit wird das Aussehen und in einigen Fällen auch die Form bestimmt. Die Werte von 0 ansteigend entsprechen der Reihenfolge der Bevel-Enumeration im Inspector – siehe auch Abbildung rechts.

Der Wert 4 = No Bevel sorgt nur unter Windows und Linux für einen BevelButton ohne Rahmen. Die Werte Rounded = 3, Round = 5, Large Round = 6 und Disclosure = 7 sind macOS-exklusiv. Unter dem aktuellen macOS (10.12) wie auch schon einigen Vorversionen unterscheiden sich einige Stile optisch nicht voneinander.

Plattformübergreifend lässt sich ein Disclosure Triangle mit dem gleichnamigen Steuerelement erzeugen.





BevelButton

Fortsetzung

Properties (Fortsetzung)

Bold As Boolean

Lässt die **Caption** (den Titel) des Buttons in **Fettschrift** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property **in der IDE** gesetzt.

ButtonType As Integer

Bestimmt die Art, wie sich der „**Gedrückt**“-Zustand des **BevelButton** verändert, wenn der Button angeklickt wurde. Die möglichen Werte entsprechen der **ButtonType**-Enumeration in der IDE:

- | | |
|---------------|---|
| 0 = Button = | Wie PushButton: Bleibt solange gedrückt, wie die Maustaste gedrückt wird. |
| 1 = Toggles = | Umschalter. Verbleibt im Zustand, bis wieder auf ihn geklickt wird. |
| 2 = Sticky = | Einschalter. Verbleibt im gedrückten Zustand. |

Caption As String

Der **Titel** des Buttons. Häufig wird diese Property **in der IDE** gesetzt. Siehe **PushButton.Caption** zum Setzen von **Accelerator Keys**.

CaptionAlign As Integer

Die Ausrichtung des Button-Titels (der **Caption**). Die möglichen Werte entsprechen der gleichnamigen Enumeration in der IDE:

- | | |
|-------------------|--|
| 0 = Flush Left = | linksbündig |
| 1 = Flush Right = | rechtsbündig |
| 2 = System = | automatisch nach Systemspracheinstellung |
| 3 = Center = | zentriert |

CaptionDelta As Integer

Der Abstand des Titels (der **Caption-Property**) zum **Icon** des Buttons. Häufig wird diese Property **in der IDE** gesetzt.

CaptionPlacement As Integer

Die Position des Titels (der **Caption**) in Relation zum **Icon**. Die möglichen Werte entsprechen der gleichnamigen Enumeration **in der IDE**:

- | | |
|------------------------|---|
| 0 = Sys Direction = | automatisch nach Systemvorgabe |
| 1 = normally = | normal. I. d. R. identisch mit Sys Direction. |
| 2 = Right of Graphic = | rechts des Icons |
| 3 = Left of Graphic = | links des Icons |
| 4 = Below Graphic = | unter dem Icon |
| 5 = Above Graphic = | über dem Icon |



Sachdienliche Hinweise zum Unterschied zwischen 0 und 1 werden gerne entgegenommen. Die Sprachreferenz hilft hier leider nicht weiter.

HasBackColor As Boolean !

 Nur wenn dieser Wert **True** ist, wird (nur unter Windows und Linux) die **BackColor** dargestellt. Meistens wird diese Property **in der IDE** gesetzt.



BevelButton

Fortsetzung

Properties (Fortsetzung)

HasMenu As Integer

Ob und wenn ja an welcher Position der BevelButton ein PopupMenu anzeigt. Die möglichen Werte entsprechen der gleichnamigen Enumeration in der IDE:

0 = No Menu	= kein Menü
1 = Normal Menu	= Menü auf der linken Seite
2 = Menu on Right	= Menü auf der rechten Seite

 Das Menü muss programmatisch via AddRow hinzugefügt werden. HasMenü sorgt dann für seine Darstellung während eines Klicks auf den BevelButton. In MenuValue kann der Index des angewählten Menüeintrags ausgelesen werden.

Icon As Picture

Ein optionales Bild, das vom BevelButton dargestellt werden kann. Kann in der IDE zugewiesen oder auch programmatisch zur Laufzeit.

 Oftmals bietet sich die letztere Methode eher an, da Icon die Größe des Bildes übernimmt. Ist dieses zu groß, kann man es vor Zuweisen der Property programmatisch skalieren.

IconAlign As Integer

Die Positionierung von Icon innerhalb des BevelButtons. Die möglichen Werte entsprechen der gleichnamigen Enumeration in der IDE:

0 = Sys Direction	= Systemstandard
1 = Center	= Mitte
2 = Left	= Mitte links

3 = Right	= Mitte rechts
4 = Top	= Oben zentriert
5 = Bottom	= Unten zentriert
6 = Top Left	= Linke obere Ecke
7 = Bottom Left	= Linke untere Ecke
8 = Top Right	= Rechte obere Ecke
9 = Bottom Right	= Linke untere Ecke

IconDx, IconDy As Integer

Ein zusätzlicher Versatz des Icons in X- oder Y-Richtung, jeweils abhängig von IconAlign. Bei IconAlign = 1 = Center werden diese Werte nicht berücksichtigt.

Italic As Boolean

Lässt die Caption (den Titel) des Buttons in Kursivschrift erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property in der IDE gesetzt.



BevelButton

Fortsetzung

Properties (Fortsetzung)

MenuValue As Integer

Der Index des vom Benutzer angewählten Menüeintrags, sofern der Button ein **PopupMenu** besitzt, bzw. der Menüeintrag, der vorausgewählt (mit Häkchen versehen) dargestellt werden soll. Vorgabewert 0. –1 löscht das Häkchen.

Der folgende Code im **Open**-Event eines PushButtons definiert ein **PopupMenu** für diesen:

```
Dim Monate() As String = Array("Januar", "Februar", "März", "April", _
    "Mai", "Juni", "Juli", "August", "September", "Oktober", "November", _
    "Dezember")
Me.Caption = "Monate"
Me.CaptionAlign = 0 // linksbündig
Me.HasMenu = 2      // Menü auf der rechten Seite
For Each m As String In Monate
    Me.AddRow(m) // Als Menüeintrag hinzufügen.
Next
```

Dieser Code im **Action**-Event gibt den gewählten Monat im Klartext aus:

```
MsgBox(Me.List(Me.MenuValue))
```

TextColor As Color

Die Farbe der **Caption** des Buttons. Vorgabe Schwarz.

TextFont As String

Der Name der Schrift, der zur Darstellung der **Caption** (des Titels) des **BevelButtons** benutzt werden soll. Siehe **PushButton/TextFont**. Häufig wird diese Property in der IDE gesetzt.

TextSize As Single

Die Größe der Schrift, die zur Darstellung der **Caption** (des Titels) des **BevelButtons** benutzt werden soll. Eingabe von 0 definiert dabei die Standardgröße. Meistens wird diese Property in der IDE gesetzt.

TextUnit As RectControl.FontUnits

Die Einheit, in der **TextSize** aufzufassen ist.

Underline As Boolean

Lässt die **Caption** (den Titel) des Buttons **unterstrichen** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property in der IDE gesetzt.

Value As Boolean

Ob der Button **gedrückt** dargestellt wird.

 Programmatisches Setzen dieses Werts auf **True** feuert nicht den **Action-Event**.



BevelButton

Fortsetzung

Methoden

AddActionNotificationReceiver (theReceiver as ActionNotificationReceiver)

Registriert einen ActionNotificationReceiver auf den Action-Event des BevelButtons. Teil des ActionNotificationReceiver-Interface.

AddRow (theText As String)

Fügt dem Menu des Buttons einen Eintrag mit Titel theText hinzu. Siehe **MenuValue**.

AddSeparator

Fügt dem Menu des Buttons einen Trennstrich hinzu. Siehe **MenuValue**.



Ein Separator zählt im String-Array der Menüeinträge mit. Er kann aber nicht ausgewählt werden.

DeleteAllRows

Löscht das Menu des BevelButtons vollständig. Es kann dann wieder mit den obigen Methoden neu aufgebaut werden.

InsertRow (row As Integer, theText As String)

Fügt dem Menu des Buttons einen Eintrag mit Titel theText an der Position row hinzu. Folgende Einträge werden um 1 verschoben. Siehe **MenuValue**.

List (row As Integer) As String

Gibt den Title des Menüeintrags an Position row zurück. Siehe **MenuValue**.

RemoveActionNotificationReceiver (theReceiver as ActionNotificationReceiver)

Entfernt die Registrierung für einen ActionNotificationReceiver auf den Action-Event des BevelButtons. Teil des ActionNotificationReceiver-Interface.

RemoveRow (row As Integer)

Entfernt den Menüeintrag an Position row aus der Liste der Menüeinträge.



Canvas

Klasse/Steuerelement

Der Canvas ist das vielseitigste Desktop-Steuerelement und das grundlegendste zugleich: Technisch gesehen ist ein Canvas ein **rechteckiger Bildschirmausschnitt** mit einer großen Palette an Events zur Benutzerinteraktion, wobei die Art der Reaktion(en) und die visuelle Darstellung des Canvas ganz in der Hand des Programmierers liegen. Die Visualisierung wird dabei meist über den Paint-Event realisiert, bei dem das **Graphics**-Objekt des Canvas pixel- resp. pointsorientiert mit den Draw-Funktionen der **Graphics**-Klasse oder vektorbasiert durch Zuhilfenahme der **Object2D**-Klassen mit Inhalt versehen werden kann.

 Auch wenn die Klassenhierarchie in Xojo etwas anderes vermuten lässt: Eigentlich sind die meisten Steuerelemente **Subklassen von Canvas**, die über viele bereits vordefinierte Eventhandler und Zeichenmethoden verfügen. Der Canvas ist die **ideale Basisklasse** zur Realisierung eigener Steuerelemente.

Events

Activate

Dieser Event feuert, wenn das Fenster des Canvas das **aktive Fenster** wird – entweder weil es geöffnet wurde oder weil es aus dem Hintergrund nach vorne gebracht wurde.

Deactivate

Das Gegenstück zu **Activate**. **Deactivate** feuert, wenn ein **anderes Fenster** desselben Programms oder ein Fenster eines anderen Programms in den **Vordergrund** geholt wird als das, auf dem der Canvas sich befindet (und das vorher das aktive Fenster war).

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

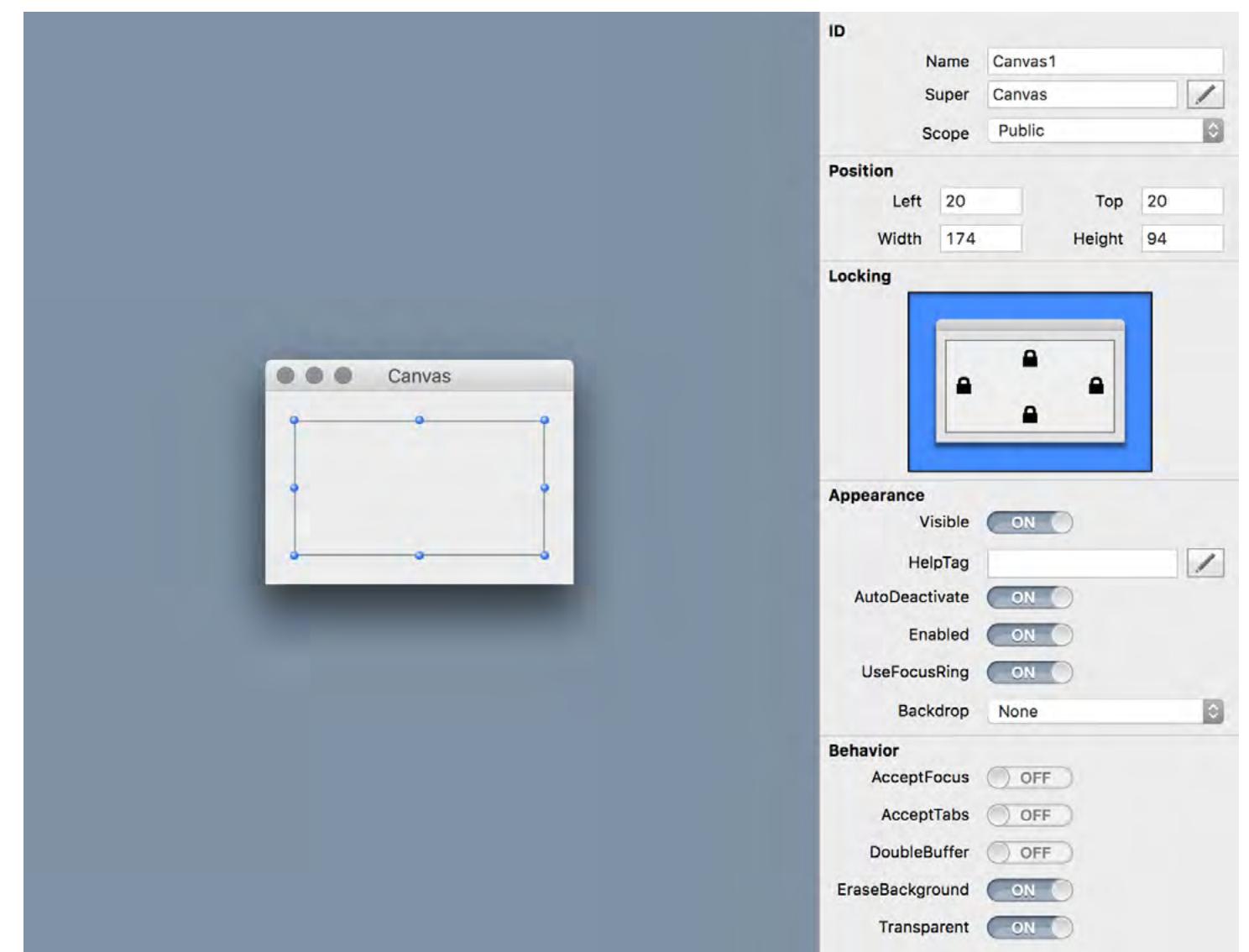
DESKTOP

PLATTFORMEN:

ALLE AUSSER IOS

SUPERKLASSE:

RECTCONTROL





Canvas

Fortsetzung

Events (Fortsetzung)

DoubleClick (X As Integer, Y As Integer)

Der Benutzer hat an den Koordinaten X,Y (relativ zur linken oberen Ecke des Canvas) einen Doppelklick ausgeführt. Feuert ungeachtet des Rückgabewerts von MouseDown.

EnableMenuItem

Ein Menubar-Menüeintrag wurde aufgerufen, während der Canvas den Focus hat. Dieser Event gibt die Gelegenheit, das Menü vor Erscheinen zu modifizieren.

GotFocus

Der Canvas hat den Focus bekommen – er wird entsprechend durch einen Rahmen markiert.

LostFocus

Der Canvas hat den Focus wieder verloren. Siehe GotFocus.

MouseDown (X As Integer, Y As Integer) As Boolean

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum Canvas selbst) gedrückt. True signalisiert, dass dieser Event selbst bearbeitet wird und keine weitere Verarbeitung durch das System mehr erfolgen soll. Infolgedessen wird der Action-Event nicht feuern. Stattdessen wird der MouseUp-Event feuern, der beim Standard-Rückgabewert False durch den Action-Event sonst nicht zum Zuge käme, sowie der MouseDrag-Event. Siehe PushButton/MouseDown.

MouseUp (X As Integer, Y As Integer) !

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum BevelButton selbst) losgelassen. Feuert nur, wenn MouseDown True zurückliefert.

MouseDrag (X As Integer, Y As Integer)

Dieser Event feuert kontinuierlich(!) innerhalb des Canvas-Gebiets, solange die Maustaste gedrückt bleibt, sich die Maus im Bereich des Canvas befindet und nur wenn True im MouseDown-Event zurückgegeben wurde. Die Parameter sind die Koordinaten der Maus relativ zur linken oberen Canvas-Ecke.

Paint (g As Graphics, Areas() As RealBasic.Rect)

Der Bildschirmbereich des Canvas oder ein oder mehrere seiner Teilbereiche müssen neu gezeichnet werden. Dies tritt etwa auf, wenn ein anderes Fenster zuvor den Canvas verdeckte, der Canvas sich öffnet oder seine Größe verändert wird.

Die Variable g ist dabei das Objekt der Graphics-Klasse, das die virtuelle Zeichenoberfläche des Canvas darstellt, und kann mit den Methoden ebenjener Klasse beschrieben werden. Im Array Areas finden sich entweder die Bereiche, die neu gezeichnet werden müssen, oder es ist leer, was andeutet, dass der komplette Canvas neu gezeichnet werden muss.

Neuzeichnen des Canvas kann auch via Invalidate oder Refresh erzwungen werden.

Der folgende Code gibt einen Text in 18 pkt Helvetica im Canvas aus, wenn er im Paint-Event steht:

```
g.Bold = True
g.Italic = True
g.TextFont = "Helvetica"
g.TextSize = 18
g.DrawString("Hallo Welt", 10, 50)
```

ScaleFactorChanged

Der ScaleFactor des Fensters, auf dem der Canvas steht, hat sich verändert, etwa weil das Fenster von einem HiDPI-Bildschirm auf einen normal auflösenden Bildschirm gezogen wurde. Ggf. sollte der Canvas neu gezeichnet werden.



Canvas

Fortsetzung

Properties

AcceptFocus As Boolean

Bestimmt, ob der Canvas den Focus erhalten kann. Kann auch in der IDE gesetzt werden.

AcceptTabs As Boolean

Wenn diese Property ihren Standardwert **False** besitzt, führt die Betätigung von TAB auf der Tastatur dazu, dass der Canvas den Focus an das nächste Steuerelement in der Tab-Reihenfolge abgibt und sein **LostFocus**-Event feuert.

Ist dieser Wert **True** und ist zugleich **AcceptFocus True**, wird auch die TAB-Taste als normaler KeyDown-Event weitergegeben, der Canvas behält den Focus. Kann auch in der IDE gesetzt werden.

Backdrop As Picture

Ein optionales Hintergrundbild, das automatisch vom Canvas hinter der im Paint-Event gezeichneten Grafik ausgegeben wird.

 Sie sollten Backdrop nicht vom Paint-Event aus zuweisen! Das führt sehr wahrscheinlich zu Grafikproblemen.

DoubleBuffer As Boolean !

Legt fest, ob der Canvas einen Bildpuffer erhält. Grafikausgaben erfolgen dann erst in den Speicher und werden nach Fertigstellung auf den Bildschirm gespiegelt, was Flackern reduziert. Kann auch in der IDE gesetzt werden.

 macOS und Linux mit GTK benutzen standardmäßig immer einen DoubleBuffer. Diese Property ist nur für Windows relevant. **EraseBackground** muss bei Verwendung **False** sein.

EraseBackground As Boolean !

Legt fest, ob der Canvas vor Zeichenoperationen gelöscht werden sollte. Vorgabewert ist **True**. Exklusiv mit DoubleBuffer zu setzen. Kann auch in der IDE gesetzt werden.

 Hat keine Auswirkungen unter macOS und Linux: Windows-exklusive Property!

Transparent As Boolean

Ob der Hintergrund des Fensters durchscheint. Vorgabewert **True**. Kann auch in der IDE gesetzt werden.

 Ein nicht-transparenter Canvas reduziert den Rechenaufwand und unter Windows das Flackern und sorgt für korrektes Clipping eingebetteter Controls unter Linux.

UseFocusRing As Boolean !

Wenn dieser Wert **False** ist, zeichnet der Canvas keinen FocusRing, auch wenn er den Focus besitzt. Standardwert **True**. Kann auch in der IDE gesetzt werden.

 Hat keine Auswirkungen unter Windows und Linux: macOS-exklusive Property!



Canvas

Fortsetzung

Methode

Scroll (DeltaX as Integer, DeltaY as Integer, [Left as Integer], [Top as Integer], [Width as Integer], [Height as Integer], [ScrollControls as Boolean = True])

Scrollt den Inhalt des Canvas um DeltaX, DeltaY, wobei optional und standardmäßig eingebettete ContainerControls und auf dem Canvas platzierte Steuerelemente mitgeschrollt werden. Soll dies nicht geschehen, muss ScrollControls False sein.

 Nicht zum Inhalt des Canvas gerechnet wird sein eventuelles Backdrop-Picture und der während des Paint-Events gezeichnete Inhalt. Letzterer kann aber scrollend aktualisiert werden, da die Scroll-Methode den Paint-Event aufruft. Dazu müssen Properties für den Scrollversatz in X- und Y-Richtung angelegt werden, die vor dem Aufruf des Scroll-Events berechnet und während des Paint-Events als X- und Y-Parameter berücksichtigt werden müssen. *Beispiel:*

Das Window, auf dem der Canvas sich befindet, besitzt zwei Integer-Properties: XScroll und YScroll. Im KeyDown-Event des Canvas werden diese berechnet und die Scroll-Methode wird aufgerufen.

```
Const PfeilLinks = 28 // Der ASCII-Code für Cursor nach links
Const PfeilRechts = 29 // Der ASCII-Code für Cursor nach rechts
Const PfeilHoch = 30 // Der ASCII-Code für Cursor nach oben
Const PfeilRunter = 31 // Der ASCII-Code für Cursor nach unten
Const ScrollSchritt = 8 // Versatz um 8 Bildschirmpts pro Tastendruck

// Je nach Richtung entsprechend scrollen
Select Case Asc(Key)
Case PfeilLinks
    XScroll = XScroll + ScrollSchritt
    me.Scroll ScrollSchritt, 0
    Return True
Case PfeilRechts
    XScroll = XScroll - ScrollSchritt
    me.Scroll ScrollSchritt, 0
    Return True
Case PfeilHoch
    YScroll = YScroll + ScrollSchritt
    me.Scroll 0, ScrollSchritt
    Return True
Case PfeilRunter
    YScroll = YScroll - ScrollSchritt
    me.Scroll 0, ScrollSchritt
    Return True
End Select
```

```
Case PfeilRechts
    XScroll = XScroll - ScrollSchritt
    me.Scroll ScrollSchritt, 0
    Return True
Case PfeilHoch
    YScroll = YScroll + ScrollSchritt
    me.Scroll 0, ScrollSchritt
    Return True
Case PfeilRunter
    YScroll = YScroll - ScrollSchritt
    me.Scroll 0, ScrollSchritt
    Return True
End Select
```

Während die eingebetteten Controls automatisch gescrollt werden, übernimmt der Paint-Event das versetzte Zeichnen des Bilds CanvasImage:

g.DrawPicture(CanvasImage, XScroll, YScroll)

 Die weiteren Parameter x, y, width und height, die optional übergeben werden können, erschließen sich mir nicht in ihrer Funktion. Meine Annahme, sie würden ein Area für den Paint-Event definieren, trifft nicht zu. Wieder einmal werden sachdienliche Hinweise gerne entgegengenommen!

 Auch wenn er scrollen kann: Der Canvas ist kein vollständiges ScrollView. Zur manuellen Hinzufügung von ScrollBars und Interaktion mit ihnen siehe das Beispielprojekt unter Graphics & Multimedia/CanvasScrolling in den Beispielen des Project Choosers.



CheckBox

Klasse/Steuerelement

Die Checkbox kombiniert ein „Abhakkästchen“ mit einem Text-Element. Sie kann drei Zustände annehmen:

Unchecked	=	ohne Häkchen
Checked	=	abgehakt
Indeterminate	=	unbestimmt

Die Caption-Property wird stets einzeilig dargestellt!

Events

Action

Feuert, wenn die Checkbox mit der Maus geklickt, ihr Accelerator Key gedrückt oder ihr Value programmatisch geändert wurde.

GotFocus

Die CheckBox hat den Focus bekommen – sie wird entsprechend durch einen Rahmen markiert.

Das funktioniert auf dem Mac nur, wenn in der Systemsteuerung/Tastatur/Kurzbefehle „Alle Steuerungen“ aktiviert ist. Siehe Pushbutton/GotFocus.

LostFocus

Die CheckBox hat den Focus wieder verloren. Siehe GotFocus.

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

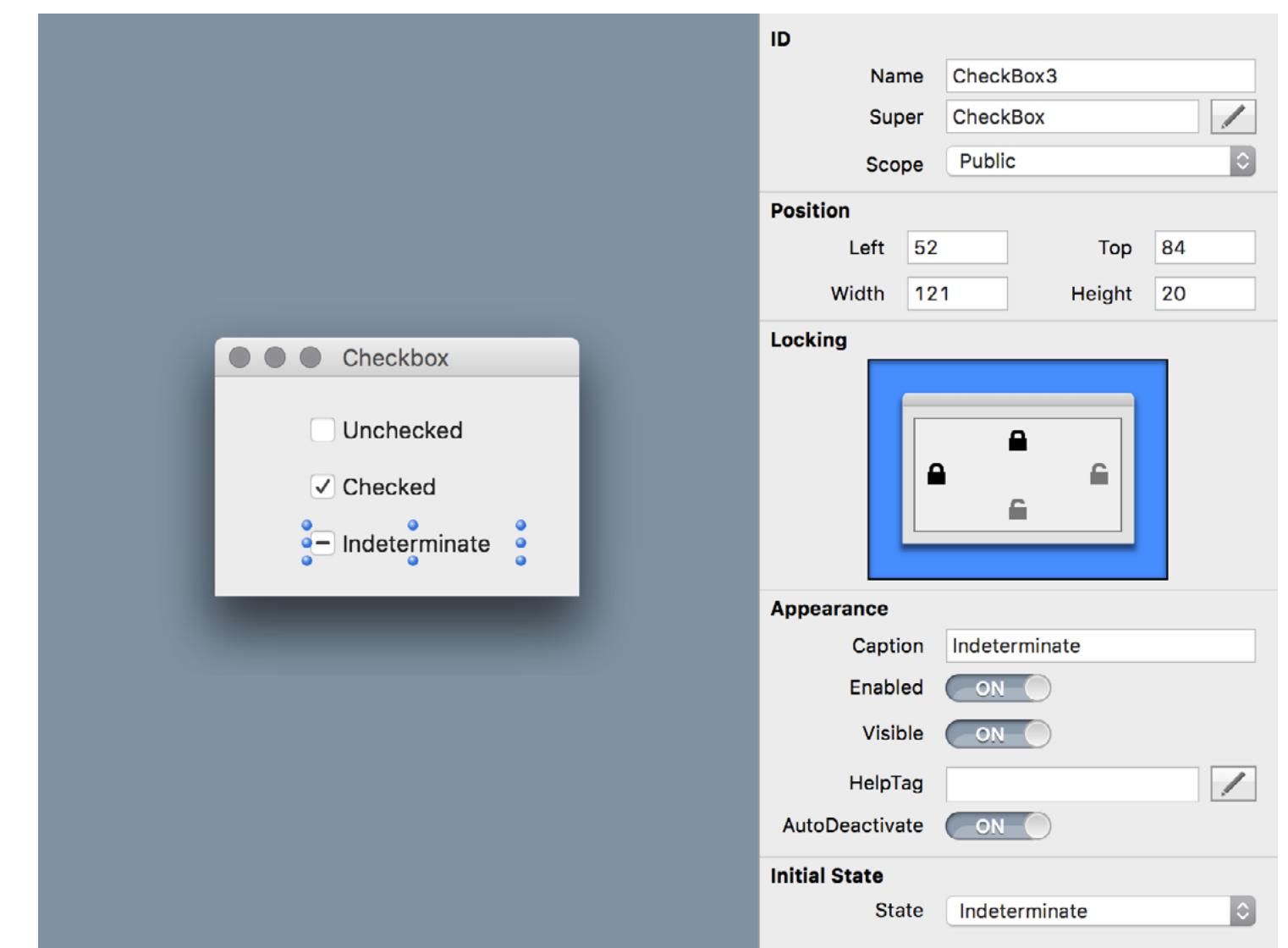
DESKTOP

PLATTFORMEN:

ALLE AUSSER IOS

SUPERKLASSE:

RECTCONTROL





CheckBox

Fortsetzung

Events (Fortsetzung)

MouseDown (X As Integer, Y As Integer) As Boolean

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zur CheckBox selbst) gedrückt. Der Rückgabewert **True** signalisiert, dass dieser Event selbst bearbeitet wird und keine weitere Verarbeitung durch das System mehr erfolgen soll. Infolgedessen wird der **Action-Event** nicht feuern und **Value** wird nicht verändert. Stattdessen wird der **MouseUp-Event** feuern, der beim Standard-Rückgabewert **False** durch den **Action-Event** sonst nicht zum Zuge käme. Siehe PushButton/**MouseDown**.

MouseUp (X As Integer, Y As Integer) !

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zur CheckBox selbst) losgelassen. Feuert nur, wenn **MouseDown True** zurückliefert.

Properties

Bold As Boolean

Lässt die **Caption** (den Titel) der CheckBox in **Fettschrift** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch **in der IDE** gesetzt werden.

Caption As String

Der **Titel** der CheckBox. Kann nur einzeilig sein. Kann auch **in der IDE** gesetzt werden. Siehe **PushButton.Caption** zum Setzen von **Accelerator Keys**.

DataField As String !

Name eines **Datenbank-Fields** in der unter **DataSource** referenzierten Tabelle. Nur relevant, falls die CheckBox in Verbindung mit einer Datenbank und einem **DataControl** verwendet wird. Kann auch **in der IDE** gesetzt werden.

DataSource As String !

Name eines **DataControls**, das eine Tabelle einer Datenbank referenziert, deren Feld unter **DataField** angegeben wurde. Nur relevant, falls die CheckBox in Verbindung mit einer Datenbank und einem **DataControl** verwendet wird. Kann auch **in der IDE** gesetzt werden. **Beispiel:**

Kopieren Sie **EddiesElectronics.sqlite** aus dem Template-Verzeichnis auf Ihre Festplatte. Legen Sie in einem neuen Desktop-Projekt in **Window1** eine Property **DB As SQLiteDatabase** an. Ziehen Sie eine **CheckBox** und ein **Data Control** auf das Fenster. Geben Sie dem **DataControl** diesen **Open-Eventhandler**:

```
Dim sourceDB As FolderItem = GetopenFolderItem("EddiesElectronics.sqlite")
If sourceDB = Nil Or sourceDB.Exists = False Then
    MsgBox "Konnte EddiesElectronics.sqlite nicht finden."
    Quit
Else
    DB = new SQLiteDatabase
    DB.DatabaseFile = sourceDB
    If Not DB.Connect then break
    me.Database = DB
    me.TableName ="customers"
    me.SQLQuery ="Select * From customers"
    me.RunQuery
End If
```



CheckBox

Fortsetzung

Properties (Fortsetzung – DataSource)

Als Handler für den Reposition-Event von DataControl1:

```
me.Caption = Trim(me.RecordSet.Field("Firstname").StringValue+" "+_
    me.RecordSet.Field("Lastname").StringValue)
```

CheckBox1 erhält als DataField in der IDE den Wert "Taxable", als DataSource "DataControl1" und die Caption "steuerabzugsfähig".

Führen Sie das Projekt jetzt aus. Lokalisieren Sie die Datenbank auf der Festplatte. Das DataControl zeigt nun einen Namen an, und Sie können mit den Tasten des Controls navigieren. Zugleich verändert CheckBox1 synchron ihren Wert analog zum Inhalt des Felds "Taxable" der Tabelle "Customers".

Italic As Boolean

Lässt die Caption (den Titel) der CheckBox in Kursivschrift erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch in der IDE gesetzt werden.

State As CheckBox.CheckedStates !

Im Gegensatz zum booleschen Value kann eine CheckBox drei Zustände annehmen: unchecked, checked und indeterminate. Erhält State einen der beiden letzteren Werte, wird Value automatisch auf True gesetzt.

TextFont As String

Der Name der Schrift, der zur Darstellung der Caption (des Titels) der CheckBox benutzt werden soll. Siehe PushButton/TextFont. Kann auch in der IDE gesetzt werden.

TextSize As Single

Die Größe der Schrift, die zur Darstellung der Caption (des Titels) des BevelButtons benutzt werden soll. Eingabe von 0 definiert dabei die Standardgröße. Kann auch in der IDE gesetzt werden.

Underline As Boolean

Lässt die Caption (den Titel) des der CheckBox unterstrichen erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch in der IDE gesetzt werden.

Value As Boolean !

Der Boolesche Wert der CheckBox. True setzt den State auf Checked und False auf Unchecked.

Methode

SetBoolean (Value As Boolean)

Eine alternative Möglichkeit, den Value der CheckBox zu setzen.

```
CheckBox1.SetBoolean (True)
```

entspricht zu 100%

```
CheckBox1.Value = True
```



ComboBox

Klasse/Steuerelement

Eine ComboBox ist eine Kombination eines TextField mit einem PopupMenu. Der Benutzer kann damit wahlweise aus einer Liste auswählen oder eigene Begriffe eingeben.

 ComboBox ist eine Subklasse von PopupMenu. Konsultieren Sie diese Klasse, um alle Features jenseits RectControl einzusehen.

 macOS begrenzt die Anzahl der darstellbaren Einträge auf maximal 15. Sie können diese Limitierung durch folgenden Declare aufheben:

```
Declare Sub setNumberOfVisibleItems Lib "AppKit" selector _  
"setNumberOfVisibleItems:" (controlHandle As Integer, count As Integer)
```

Und diese Methode aufrufen durch

```
setNumberOfVisibleItems (PopupMenu1.Handle, 30) // oder anderes Limit
```

Events (siehe auch PopupMenu)

TextChanged

Der ComboBox-Text hat sich geändert – durch Auswahl aus der Liste oder durch Benutzereingabe, aber nicht durch programmatisches Setzen des ListIndex!

FRAMEWORK:

CLASSIC

PROJEKTARTEN:

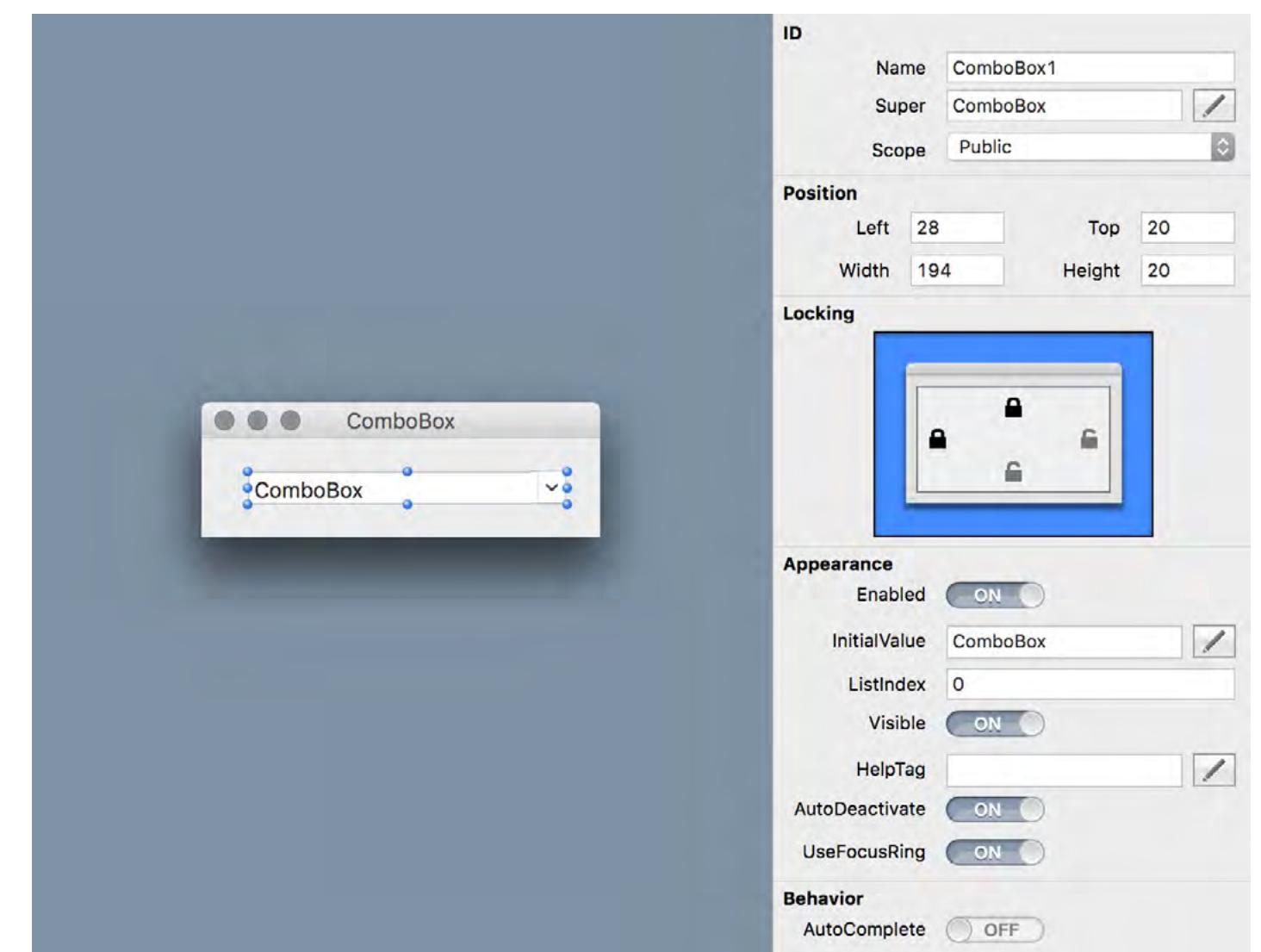
DESKTOP

PLATTFORMEN:

ALLE AUSSER IOS

SUPERKLASSE:

POPPUPMENU





ComboBox

Fortsetzung

Properties (siehe auch PopupMenu)

AutoComplete As Boolean

Schaltet das AutoComplete-Feature bei der Texteingabe ein oder aus. Ist dieser Wert **True** (Standard ist **False**), werden bei der manuellen Texteingabe **Ergänzungsvorschläge** basierend auf der aktuellen **Auswahlliste** eingeblendet.

SelLength As Integer

Lässt sich als **SelectionLength** lesen – die **Anzahl** der in der ComboBox ausgewählten (hinterlegt dargestellten) Zeichen. 0 = nur der Cursor ist gesetzt, kein Zeichen ausgewählt. Vgl. **SelStart**.

SelStart As Integer

Lässt sich als **SelectionStart** lesen – die **Position** des Einfügepunkts (**Cursors**) im **Text** der ComboBox. *Beispiel:*

```
ComboBox1.SelStart = 5
ComboBox1.SelLength = 3
```

hinterlegt auf der Abbildung der Vorseite den Wortbestandteil "Box".

Text As String

Der **Text** der **ComboBox**. Kann durch Auswahl aus der Liste oder manuelle Eingabe sowie programmatisch geändert werden, im Gegensatz zur **PopupMenu**-Klasse, bei der **Text** nur gelesen werden kann, nicht geschrieben. Kann auch in der **IDE** über **InitialValue** gesetzt werden.

UseFocusRing As Boolean !

Wenn diese Property **True** ist (der Standard), wird ein **FocusRing** um das Steuerelement gezeichnet, wenn dieses den **Focus** hat.



Nicht unter **Windows**. Dort wird der **Focus** nur über die **Einfügemarkie** im Textfeld angezeigt.

Methoden siehe PopupMenu



PopupMenu

Klasse/Steuerelement

Eine PopupMenu stellt auf Anklicken eine Liste dar, aus der der Benutzer eine Auswahl treffen kann. Man könnte ein PopupMenu auch als Steuerelement zur ausschließlichen Darstellung von Kontextmenüs ansehen. Die Tochterklasse ComboBox ergänzt das PoupMenu um die Möglichkeit, benutzerdefinierte Einträge entgegenzunehmen.

Events

 Kontextmenüs und ihre dazugehörigen Events sind beim PopupMenu unter macOS nicht darstell- bzw. nutzbar.

Change

Die Auswahl des PopupMenus wurde verändert. Sie kann über die Property ListIndex bestimmt werden.

GotFocus

Das PopupMenu hat den Focus bekommen – er wird entsprechend durch einen Rahmen markiert.

 Das funktioniert auf dem Mac nur, wenn in der Systemsteuerung/Tastatur/Kurzbefehle „Alle Steuerungen“ aktiviert ist. Siehe PushButton/GotFocus.

LostFocus

Das PopupMenu hat den Focus wieder verloren. Siehe GotFocus.

FRAMEWORK:

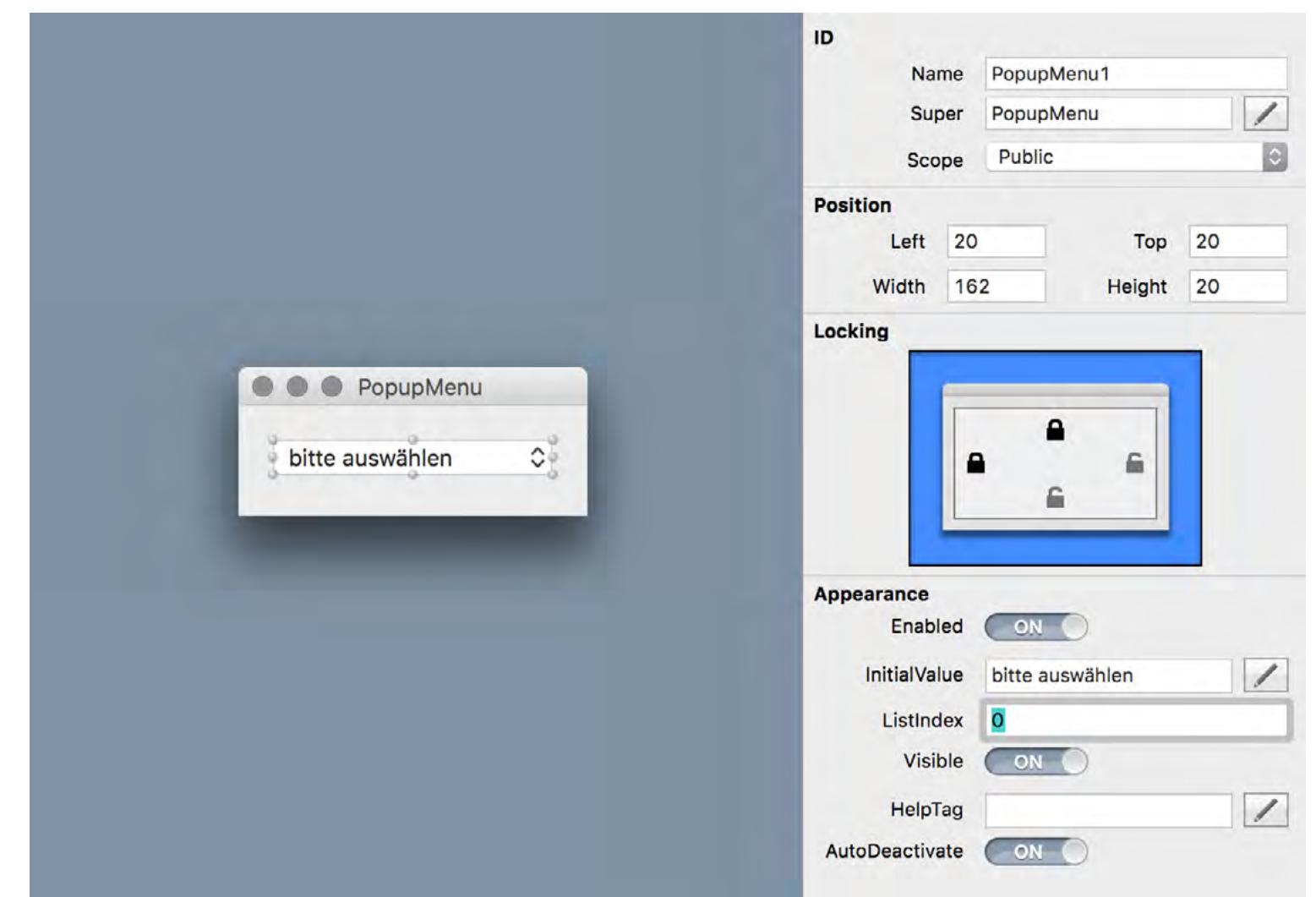
CLASSIC

PROJEKTARTEN:

DESKTOP

PLATTFORMEN: **ALLE AUSSER IOS**

SUPERKLASSE: **RECTCONTROL**





PopupMenu

Klasse/Steuerelement

Events (Fortsetzung)

MouseDown (X As Integer, Y As Integer) As Boolean

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum PushButton selbst) gedrückt. Der Rückgabewert **True** signalisiert, dass dieser Event selbst bearbeitet wird und keine weitere Verarbeitung durch das System mehr erfolgen soll. Infolgedessen wird der **Action-Event** nicht feuern. Stattdessen wird der **MouseUp-Event** feuern, der beim Standard-Rückgabewert **False** durch den **Action-Event** sonst nicht zum Zuge käme.

MouseUp (X As Integer, Y As Integer) !

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zur CheckBox selbst) losgelassen. Feuert nur, wenn **MouseDown True** zurückliefert.

 **MouseDown** und **MouseUp** sind unter Linux nicht nutzbar.

Properties

 Height verändert nicht die Höhe des PopupMenus unter macOS.

Bold As Boolean

Lässt den Text des PopupMenus in Fettschrift erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch in der IDE gesetzt werden.

DataField As String !

Name eines Datenbank-Fields in der unter **DataSource** referenzierten Tabelle. Nur relevant, falls das PopupMenu in Verbindung mit einer Datenbank und einem **DataControl** verwendet wird. Kann auch in der IDE gesetzt werden.

DataSource As String !

Name eines **DataControls**, das eine Tabelle einer Datenbank referenziert, deren Feld unter **DataField** angegeben wurde. Nur relevant, falls das PopupMenu in Verbindung mit einer Datenbank und einem **DataControl** verwendet wird. Kann auch in der IDE gesetzt werden. Siehe **CheckBox.DataSource**.

InitialValue() As String ☐

Die Einträge, die das PopupMenu als **Auswahlmöglichkeiten** anbieten soll. Kann nur in der IDE als **Return-separierte Liste** gesetzt werden. Programmatisch werden die Werte mittels **AddRow** oder **AddRows** definiert. Der anzuzeigende Wert kann mittels **ListIndex** bestimmt werden.

Italic As Boolean

Lässt den Text des PopupMenus in Kursivschrift erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch in der IDE gesetzt werden.

ListCount As Integer ☐

Die Anzahl der Einträge in der Liste der **Auswahlmöglichkeiten**.



PopupMenu

Klasse/Steuerelement

Properties (Fortsetzung)

ListIndex As Integer

Die Position des anzuzeigenden bzw. angezeigten Eintrags in der Liste der Auswahlmöglichkeiten. Erstes Element = 0.

Text As String

Der Text des PopupMenus. Kann nur über Auswahl aus der Liste geändert werden.

TextFont As String

Der Name der Schrift, der zur Darstellung der Text-Property des PopupMenus benutzt werden soll. Siehe PushButton.TextFont. Kann auch in der IDE gesetzt werden.

TextSize As Single

Die Größe der Schrift, die zur Darstellung der Caption (des Titels) des PopupMenus benutzt werden soll. Eingabe von 0 definiert dabei die Standardgröße. Meistens wird diese Property in der IDE gesetzt. Kann auch in der IDE gesetzt werden.

Underline As Boolean

Lässt den Text des PopupMenus unterstrichen erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Kann auch in der IDE gesetzt werden.

Methoden

AddRow (Item As String)

Fügt einen neuen Eintrag namens Item am Ende der Auswahlliste hinzu.

 "--" fügt unter macOS einen Separator ein, nicht jedoch unter Windows und Linux, wo ein anwählbarer Eintrag namens "--" erzeugt wird.

AddRows (Items() As String)

Fügt neue Einträge mit entsprechenden Namen in Reihenfolge des Arrays am Ende der Auswahlliste hinzu.

```
PopupMenu1.AddRows (Array("Frühling", "Sommer", "Herbst", "Winter"))
```

AddSeparator

Hängt an das Ende der Auswahlliste unter macOS einen nichtanwählbaren Trennstrich-Eintrag an.

 Nur unter macOS, nicht jedoch unter Windows und Linux, wo ein anwählbarer Eintrag namens "--" erzeugt wird.

DeleteAllRows

Löscht die gesamte Auswahlliste.



PopupMenu

Klasse/Steuerelement

Methoden (Fortsetzung)

InsertRow (RowIndex As Integer, Item As String)

Fügt einen neuen Eintrag mit dem Namen von Item an Position RowIndex in die Auswahlliste ein. Verschiebt folgende Einträge entsprechend.

```
PopupMenu1.InsertRow (0, "Karneval")
```

List (Index As Integer) As String

Liefert den Eintrag aus der Auwahlliste an Position Index. Für das obige Beispiel:

```
Dim Auswahl As String = PopupMenu1.List(0) // -> "Karneval"
```

RemoveRow (Index As Integer)

Entfernt den Eintrag an Position Index der Eintragsliste. Nummerierung beginnt mit 0.

RowTag (Index As Integer) [Assigns Value] As Variant

Eine Identifikationsmarke oder zusätzliche Daten, die einem Eintrag in der Auswahlliste hinzugefügt werden können. *Beispiel:*

Kontakt ist ein (macOS-exklusiver) AddressBookContact. Beim Erstellen der PopupMenu-Auswahlliste:

```
PopupMenu1.DeleteAllRows
PopupMenu1.AddRow (trim(Kontakt.Firstname+" "+Kontakt.LastName))
PopupMenu1.RowTag(0) = Kontakt
```

Beim Auswerten der Auswahlveränderung kann dann auf den Kontakt zurückgegriffen werden. Im Change-Eventhandler von PopupMenu1 daher:

```
Dim Index As Integer = me.ListIndex
If Index > -1 then
    Dim Kontakt As AddressBookContact = me.Rowtag (Index)
End If
```

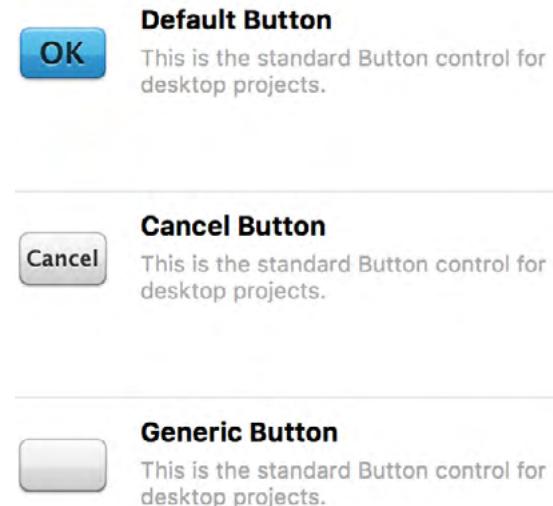


PushButton

Klasse/Steuerelement

Ein PushButton ist die übliche Schaltfläche in Form eines meist mit Text beschrifteten Knopfes, der eigentlich ein Taster ist: Wird er „gedrückt“, feuert sein Action-Event. Sein Verhalten und Aussehen können dabei – vor allem auf macOS – auf vielfache Weise modifiziert werden.

- In der Library finden Sie den PushButton gleich dreifach:



Lassen Sie sich nicht davon irritieren, dass diese Klassen nicht in der Sprachreferenz gelistet sind. Es sind drei Varianten des PushButton, wobei die Property **Default** beim **Default Button** (Vorgabeknopf) auf True gesetzt ist und die Property **Cancel** beim **Cancel Button**. Der **Generic Button** (allgemeiner, normaler Button) entspricht einem PushButton, dessen **Default** und **Cancel**-Property False sind. Sie dienen nur dem schnelleren Layouten. Auf einem Window platziert ergeben Sie stinknormale PushButtons – nur eben mit vorselektierten Properties.

- Unter Windows und Linux kann die Caption-Property des Buttons einen Accelerator Key beinhalten – ein Zeichen, das, wenn es (zusammen mit Alt) auf der Tastatur gedrückt wird, den Button auslöst: "&Accelerator" als Caption lässt den Button auf "A" reagieren.

Wollen Sie ein kaumännisches „Und“ in der Caption verwenden, müssen Sie dafür zwei hintereinander eingeben:

```
meinButton.Caption = "Speichern && beenden"
```

- Für den Mac benötigen Sie entweder einen Declare, oder Sie verwenden den KeyDown-Event, bei dem Sie auch gleich den Status von Sondertasten abfragen können, um ggf. die Push-Methode aufzurufen.

FRAMEWORK:

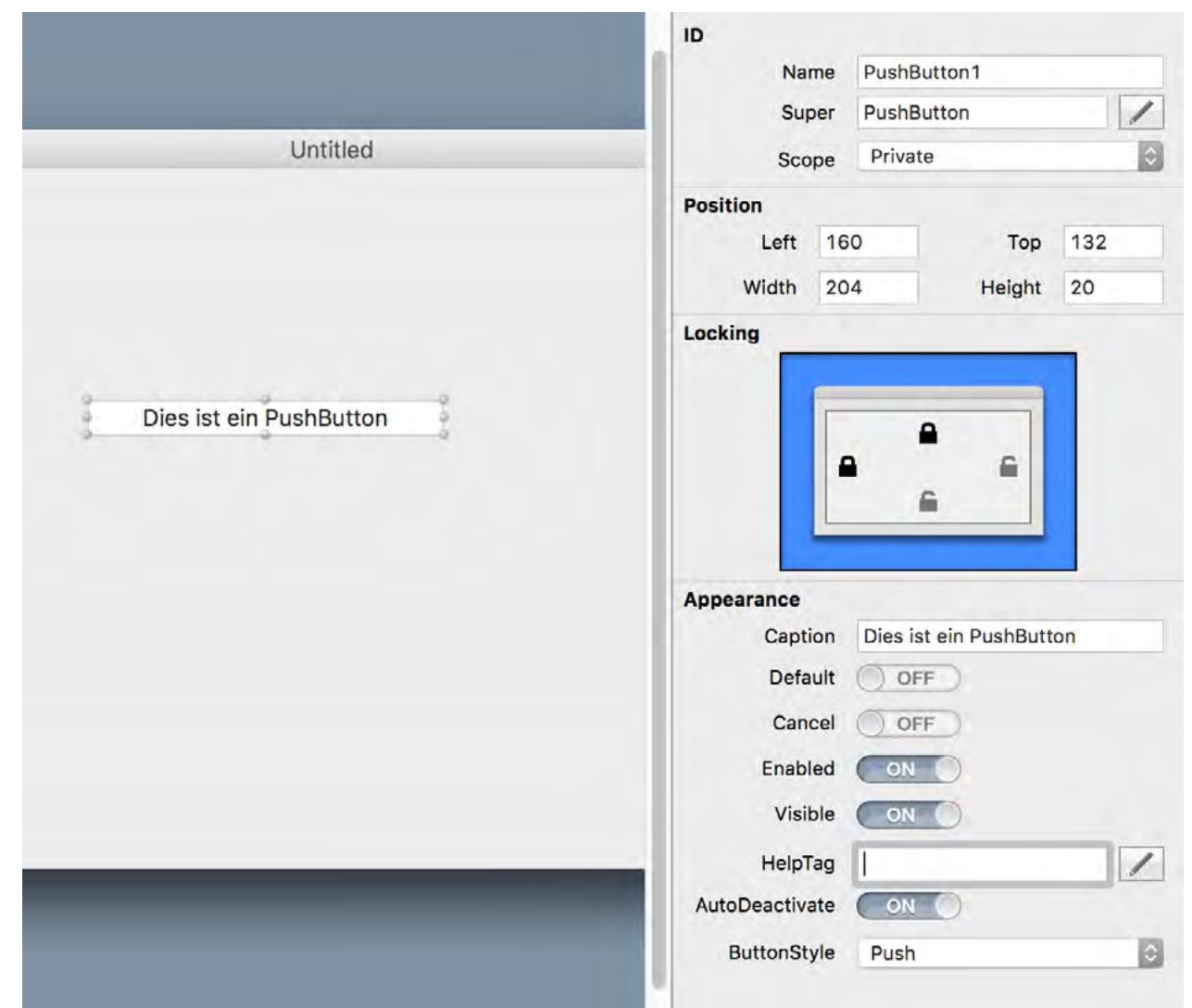
CLASSIC

PROJEKTARTEN:

DESKTOP

PLATTFORMEN: **ALLE AUSSER IOS**

SUPERKLASSE: **RECTCONTROL**





PushButton

Fortsetzung

Events

Action

Feuert, wenn der Button mit der Maus geklickt, seine Push-Methode benutzt oder sein Accelerator Key gedrückt wurde.

GotFocus

Der Button hat den Focus bekommen – er wird entsprechend durch einen Rahmen markiert.

 Das funktioniert auf dem Mac nur, wenn in der Systemsteuerung/Tastatur/Kurzbefehle „Alle Steuerungen“ aktiviert ist.



LostFocus

Der Button hat den Focus wieder verloren. Siehe GotFocus.

MouseDown (X As Integer, Y As Integer) As Boolean

Ein Maus-Button wurde an den Koordinaten X und Y (relativ zum PushButton selbst) gedrückt.

Der Rückgabewert True signalisiert, dass dieser Event selbst bearbeitet wird und keine weitere Verarbeitung durch das System mehr erfolgen soll. Infolgedessen wird der Action-Event nicht feuern. Stattdessen wird der MouseUp-Event feuern, der beim Standard-Rückgabewert False durch den Action-Event sonst nicht zum Zuge käme.

 MouseDown besagt nicht, dass die linke Maustaste gedrückt wurde. Ein MouseDown-Eventhandler wie

```
If IsContextualClick Then
    Dim m As MenuItem = EditMenu.Clone.Popup
    Return True
Else
    Return False
End If
#Pragma Unused X
#Pragma Unused Y
```

lässt bei einem Rechtsklick auf den Button das Edit-Menü erscheinen und löst den Action-Event nicht aus, während letzterer bei einem Linksklick getriggert wird.



PushButton

Fortsetzung

Properties

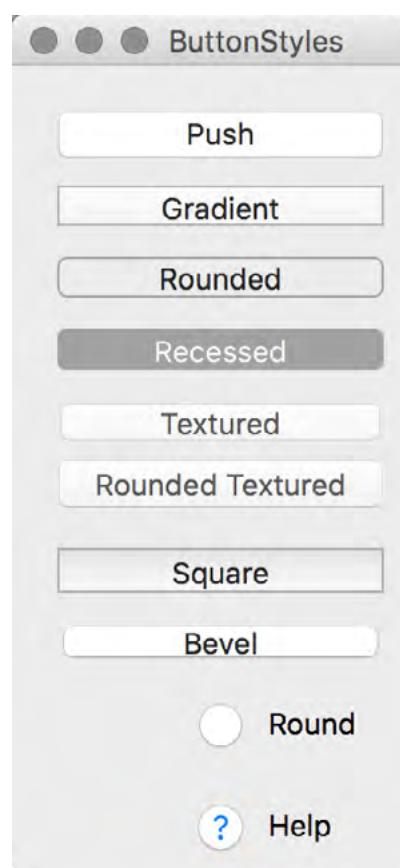
Bold As Boolean

Lässt die **Caption** (den Titel) des Buttons in **Fettschrift** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property **in der IDE** gesetzt.

ButtonStyle As Pushbutton.ButtonStyle

 Nur unter **MacOS** verfügbar!

Legt das **Aussehen** des Buttons aus der Enumeration **ButtonStyle** fest. **Kann nur in der IDE** gesetzt werden.



 Einige **ButtonStyles** lassen auch andere Höhen als die Standard-Höhe zu, und einige zeichnen etwas versetzt gegenüber dem Rahmen in der IDE. Verlassen Sie sich auf die Vorschau im Layout-Editor!

Cancel As Boolean

Wenn diese Property **True** ist, lösen die **Tastatureingaben** **ESC** und **CMD-. (Punkt)** den **Action-Event** des **PushButtons** aus.

Meistens wird diese Property **in der IDE** oder durch Ziehen des **Cancel Buttons** auf das Layout gesetzt.

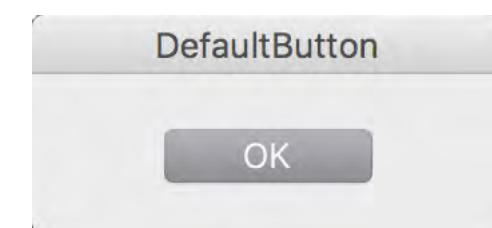
Caption As String

Der **Titel** des Buttons. Häufig wird diese Property **in der IDE** gesetzt.

Default As Boolean

Wenn diese Property **True** ist, wird der **PushButton** als **Vorgabe-Button** hervorgehoben dargestellt und Return sowie Enter auf der Tastatur lösen seinen Action-Event aus.

Meistens wird diese Property **in der IDE** oder durch Ziehen des **Default Buttons** auf das Layout gesetzt.



Italic As Boolean

Lässt die **Caption** (den Titel) des Buttons in **Kursivschrift** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property **in der IDE** gesetzt.



PushButton

Fortsetzung

Properties (Fortsetzung)

TextFont As String

Der Name der Schrift, der zur Darstellung der **Caption** (des Titels) des PushButtons benutzt werden soll. Die Systemschrift ist dabei als "System" und die kleinere Systemschrift als "SmallSystem" verfügbar. Liefert das verwendete Betriebssystem keine kleine Systemschrift, wird stattdessen die normale verwendet; ebenso, wenn die gewählte Schrift nicht auf dem System verfügbar ist. Häufig wird diese Property in der **IDE** gesetzt.

```
meinButton.TextFont = "Helvetica"
```

TextSize As Single

Die Größe der Schrift, die zur Darstellung der **Caption** (des Titels) des PushButtons benutzt werden soll. Eingabe von 0 definiert dabei die Standardgröße. Meistens wird diese Property in der **IDE** gesetzt.

Underline As Boolean

Lässt die **Caption** (den Titel) des Buttons **unterstrichen** erscheinen, sofern die verwendete Schrift einen solchen Schnitt beinhaltet. Meistens wird diese Property in der **IDE** gesetzt.

Methoden

AddActionNotificationReceiver (theReceiver as ActionNotificationReceiver)

Registriert einen **ActionNotificationReceiver** auf den **Action-Event** des PushButtons. Teil des **ActionNotificationReceiver**-Interface.

Push

Simuliert einen **Mausklick** auf den Button (auch optisch) und löst seinen **Action-Event** aus.

RemoveActionNotificationReceiver (theReceiver as ActionNotificationReceiver)

Entfernt die Registrierung für einen **ActionNotificationReceiver** auf den **Action-Event** des PushButtons. Teil des **ActionNotificationReceiver**-Interface.



Index

Symbole

- Definition 34

, Definition 46

*

Definition 34

/

Definition 34

//

Definition 46

+

bei String 131, 134

Definition 34

mit Zeichenketten 41

< 128

Definition 36

<=

Definition 36

<>

Definition 36

= 128

Definition 36

> 128

Definition 36

>=

Definition 36

32 Bit 27

64 Bit 27

&c 118

&h 118

.Net 11

A

Abakus 24–26

Accelerator Key
Button
Referenz 176

AcceptFileDrop
Referenz 154

AcceptFocus
Property
Referenz

BevelButton 158

Canvas 165

AcceptPictureDrop
Referenz 155

AcceptRawDataDrop
Referenz 155

AcceptTabs
Property
Referenz

Canvas 165

AcceptTextDrop
Referenz 155

Action
Event 92
Referenz

BevelButton 158

CheckBox 167

PushButton 177

Action Event 151, 176, 177

Add 19

Action-Event 21

actionNotificationReceiver
Interface

Definition 91

actionSource
Interface 91, 92

ActionSourceInterfaceExample 91

Activate
Event

Referenz

Canvas 163

Active
Referenz 153

Adams, Douglas 17

AddActionNotificationReceiver
Definition 91

Referenz 179

AddHandler
Definition 89

Referenz 106

Addition 34
von Strings 131, 134

AddressOf
Definition 90

Referenz 107

AddRow
Methode
Referenz

BevelButton 162

PopupMenu 174

AddRows
Methode
Referenz

PopupMenu 174

AddSeparator
Methode
Referenz

BevelButton 162

PopupMenu 174

Aggregates
Definition 88

Alpha 118

Anweisung
bedingte 35

Append
Referenz 111

Application 65

Application Identifier 15

Application Name 15

App-Objekt
Definition 30, 65

ARC
Definition 99

Array
mehrdimensional

Referenz 113

Referenz 111

As
Definition 27

Referenz 108, 178, 179

Asc 131

ASCII 37

Auto 143
Referenz 114

Autocomplete 11
Definition 32

AutoComplete
Property

Referenz

ComboBox 171
Referenz

BevelButton 153

Automatic Reference Counting.
Siehe ARC

B

Back 16

BackColor

Property

Referenz

BevelButton 158

Backdrop

Property

Referenz

Canvas 165

Bad Data Exception 121, 126

Barry, Andrew 10

Basic
Definition 10

Bedingung 35

BeginsWith 135

Bevel

Property

Referenz

BevelButton 158

BevelButton

Hintergrundfarbe 158

Referenz 157

BigEndian 133

Bildschirm-Neuaufbau 155

Binärlogik

Definition 24

Bit
Definition 24

Black 119

Blue 118, 119

Bluetooth 87

Bold

Property

Referenz

BevelButton 159
Referenz

CheckBox 168, 173

PushButton 178

Boolean 128
Definition 39

Referenz 115

Boole, George 39

Break
Definition 60

Breakpoint
Definition 60

Brown 119

Bug
Definition 33

Button. Siehe PushButton

ButtonStyle
Referenz 178

ButtonType
Property

Referenz

BevelButton 159

Byref 139

ByRef 133
Definition 98

by Reference. Siehe byRef

Byte
Definition 26

Zahlenraum 26

ByteValue 133

by Value. Siehe byVal

C

Cancel
Property 176

Referenz 178

Cancel Button
Referenz 176

Cannot assign a value to this property

73

Canvas 91, 92

Referenz 163

Caption 19
Property

Referenz

BevelButton 159
Referenz

CheckBox 168

PushButton 178

CaptionAlign
Property

Referenz

BevelButton 159

CaptionDelta
Property

Referenz

BevelButton 159

CaptionPlacement
Property

Referenz

BevelButton 159

CGFloat
Referenz 117

Change
Event

Referenz

PopupMenu 172

Characters 135

CheckBox
Referenz 167

Class 128. Siehe Klasse

Classic
Framework 134

Console
Referenz 145

Desktop
Referenz 146

Class Interface. Siehe Interface

Clear 119

Clone 97

Close 149
Event

Referenz 149

CMY 119

Code
deaktivieren 46

Code-Editor 27



Referenz 118	CheckBox 168 PopupMenu 173	Destructor Referenz 140	Else Definition 35	Feuern. Siehe Event	BevelButton 158
ComboBox	Referenz 170	DataSource Property Referenz CheckBox 168 PopupMenu 173	Dialogbox. Siehe MsgBox	FileType 154, 155	Canvas 164
Company Name 15		Dictionary 141	Enabled Property Referenz 153	FileType Sets Editor 154, 155	CheckBox 167
Compare 135		Dim bei Objekten 55	Encoding 131	Flackern Bildschirmaufbau reduzieren 165	PopupMenu 172
CompareCaseSensitive 135		Definition 27	End Function. Siehe Function	Fließkommadatentypen 28	PushButton 177
Computed Property 75. Siehe Property: Computed		Referenz 109	Endianness Definition 133	Focus 158, 164, 167, 172, 177	Gray 119
Connect	Definition 89	dirty 155	End If Definition 35	Referenz 156	Green 118, 119
Connected	Event 89	Disclosure Triangle 158	End Sub. Siehe Sub	setzen. Siehe SetFocus	Großschreibung. Siehe Kontextsensitivität
Console 15, 16	Beispielprojekt 29 Definition 29	Division durch 0 35	EndsWith 136	FocusRing 171	GtkWidget 149
ConstructContextMenu	Referenz 151	Do ... Loop Definition 39	Entwicklerevangelist Definition 3	FontUnits Enumeration Referenz RectControl 156	H
Constructor 67, 79	Definition 62 Referenz 140	Dotnotation. Siehe Punktnotation	Entwicklungssystem Definition 3, 7	Handle 149	
ContextMenu 151		Double 128 Definition 28 Referenz 123	Entwicklungsumgebung Integrierte. Siehe IDE	Definition 97	
ContextMenuItemAction	Referenz 151	DoubleBuffer Property Referenz Canvas 165	Equals 123, 129	Property	
Control. Siehe Steuerelement	Definition 57 Referenz 149	DoubleClick Event Referenz Canvas 164	EraseBackground Property Referenz Canvas 165	Referenz	
Control Set 149, 150		DownTo. Siehe For ... Next	ESC 178	RectControl 149	
ConvertEncoding 131		Drag & Drop 151, 154, 155	Ethernet 87	HasMenu	
Convert to Computed Property 84		DragEnter Referenz 151	Event 86 Create Event Definition from Event 86	Property	
Create Event Definition from Event 86		DragExit Referenz 152	Definition 21, 85	Referenz	
CrossBasic 10		DragOver Referenz 152	feuern 30	BevelButton 160, 161	
CString 128	Referenz 120	Deklaration implizit 41	Eventhandler 30 Definition 84	I	
CType 114	Definition 84	Delegate 112 Definition 90 Referenz 122	hinzufügen 57	Icon	
Currency 128	Referenz 121	DeleteAllRows Methode Referenz BevelButton 162 PopupMenu 174	Examples 15	Property	
Cyan 118, 119		Description 86 Definition 46	Exception 58 Definition 59	Referenz	
D		Desktop 15	NilObjectException 70 Definition 59	BevelButton 160	
DarkGray 119		E	F	I	
DataControl 168, 173	Anwendungsbeispiel 168	EddiesElectronics 168	Fakultät 11	IconAlign	
DataField	Property Referenz	Edit-Menü Comment 46 Uncomment 46	False. Siehe Boolean Referenz 115	Property	
Description 86	Definition 46	Eigenschaft. Siehe Property	Faulheit. Siehe Programmierertugend: Faulheit	Referenz	
Desktop 15		Else Definition 35	Fettschrift 159, 168, 173, 178	BevelButton 160	
				IconDx	
				Property	
				Referenz	
				BevelButton 160	
				IconDy	
				Property	
				Referenz	



BevelButton 160	Referenz 125 unsigned 27	L Leerzeichen 138 Leerzeilen im Programmcode 40	Maus Bewegung 151 Form des Mauszeigers 153 innerhalb RectControl Referenz 152	Referenz BevelButton 158 Canvas 164 CheckBox 168 PopupMenu 173	Siehe OOP Objektorientiertheit Definition 7–8
IDE 16	Definition 7	I Interface Definition 87	Left 136 Property Referenz 153	Klick 151 Position 149 Scrollrad 152	OOP 57 Definition 52 Verkapselung 67
If	Definition 35	Invalidate 91 Referenz 155	Length 136	Memory Leak 90 Definition 100	Open 149 Event Referenz 149
ternär	Definition 86	Invoke 122	LightGray 119	MenuItem 151	MouseY Referenz 149
If...End If	einzelige Form 45	iOS 15	LinksKlick 177	MenuValue Property Referenz BevelButton 161	Open-Event 30 Orange 119
IllegalCastException	Definition 80	IsA Definition 80	List Methode Referenz PopupMenu 175	Methode Definition 42–43 einfügen 43 Parameters 48 Shared Performance 77 überschreiben 81	OSType Referenz 127
immutable 134		IsContextualClick 177	ListCount Property Referenz PopupMenu 173	Mid 136	P
ImplicitInstance	Property 78	Italic Property Referenz BevelButton 160 CheckBox 169 PopupMenu 173 PushButton 178	ListIndex Property Referenz PopupMenu 174	Mod Definition 42	Paint 91 Event 155
Include "#pragma error" in the source of each method	Definition 89	J	Join 134, 138	Modulo. Siehe Mod	Referenz Canvas 164
Index	Referenz 149	K	Key 152	MouseCursor Property Referenz 153	PanelIndex Referenz 150
IndexOf 136, 137		KeyDown 176 Referenz 152	LittleEndian 133	MouseDown 152 Event 92	Parameterübergabe per Referenz. Siehe byRef per Wert. Siehe byVal
Inf 123, 129	Definition 35	KeyUp Referenz 152	LockBottom Property Referenz 153	Referenz BevelButton 158, 164, 168, 173 Canvas 164 CheckBox 168 PopupMenu 173 PushButton 177	Parent Property Referenz 153
InitialValue	Property Referenz PopupMenu 173	Klassenmethode. Siehe Methode: Shared	LockLeft Property Referenz 153	MouseDrag Event	Parse 124, 126, 128, 130 Definition 38
Input	Definition 37	Klasse Definition 55 einfügen 65 Referenz 140	LockRight Property Referenz 153	Referenz Canvas 164	PerformAction 91
Insert 16	Referenz 112	Klassenmethode 140	LockTop Property Referenz 153	MouseEnter Referenz 152	Plattformübergreifend Definition 9
InsertRow	Methode Referenz BevelButton 162 PopupMenu 175	Klassenproperty 140. Siehe Shared Property	Loop. Siehe Do ... Loop	MouseExit Referenz 152	Pop Referenz 111, 118
Installation 13		Kleinschreibung. Siehe Kontextsensitivität	LostFocus Event	MouseMove Referenz 152	PopupMenu BevelButton Referenz 161 Referenz 172
Instanz 140	Definition 56	Kochrezeptmetapher 7–8	Referenz BevelButton 158 Canvas 164 CheckBox 167 PopupMenu 172 PushButton 177	MouseUp Event	Potenz Definition 25
Instanziierung 56	Referenz 140	Kommandozeile. Siehe Console	Lowercase 136		Print Definition 32
Int8 28, 125, 128		Kommentar Definition 46			Programm Definition 24
Int16 28, 125, 128		Kontextmenü. Siehe ContextMenu			Programmieren Definition 31
Int32 28, 125, 128		Kontextsensitivität 40			
Int64 28, 125, 128		Kursivschrift 160, 169, 173, 178			
Integer 128	Datenbereiche 28, 125 Größe 27	M	Magenta 118, 119		



P	Programmierertugend Faulheit 75	Referenz 118	For ... Next 40 While ... Wend 40 Zählschleife 40	Single 128 Definition 28 Referenz 129	T	This property shadows one already defined by Compilerwarnung Definition 82
Programmierung objektorientierte. Siehe OOP	Referenz Definition 64 schwach. Siehe WeakRef			Size 133 Sondertasten 176	TabIndex	Referenz 154
Programmschleife. Siehe Schleife	Referenzzählung. Siehe ARC		Schloss-Symbol Inspector 153	Sort 112	TabStop	Property Referenz 154
Project analysieren. Siehe Analyze Project	Refresh Referenz 155		Schnellentwicklungssystem. Siehe RAD	SortWith 113	Tastatur	To. Siehe For ... Next
Project Chooser	RefreshRect Referenz 155		Schriftart 161, 169, 174, 179	Spaghetticode 10, 42	Taste 152	ToBinary 125
Property	REM Definition 46		Schriftgröße 161, 169, 174, 179	Speicherleck. Siehe Memory leak	TCPSocket 87	ToCString 120, 138
Computed Definition 73	Remove Referenz 111		Scope 140 Definition 48 privat 67 Private 68	Speicherverwaltung dynamisch 48	Teal 119	ToHex 125
Definition 54	RemoveActionNotificationReceiver Definition 92		Property Referenz 150 Public 68	Split 137	Templates 15	ToOctal 125
einfügen 66	RemoveHandler Definition 90		Screen-Redraw. Siehe Bildschirm-Neuaufbau	SQLDate Time. Siehe Date	Text 32, 37, 38, 128, 131	Top Property Referenz 153
shadowed Definition 82	RemoveRow Methode Referenz		Scroll Methode Referenz Canvas 166	Stack 74	Konstante 134	ToText 121, 123, 126, 129, 130, 131 Definition 32
Shared Definition 75	Replace 137		ScrollBar Canvas 166	Stapel. Siehe Stack	Property Referenz ComboBox 171 PopupMenu 174	Transparent Property Referenz Canvas 165
Prozedurales Programmieren 52 Definition 42	ReplaceAll 137		ScrollView 166	State 134	Referenz 134	Trim 138
Ptr 128 Definition 97	Reservierte Worte Definition 105		Select Case 141	Steuerelement. Siehe Control	TextChanged Event Referenz ComboBox 170, 171	TrimLeft 138
Punktnotation 134 Definition 32	Self Definition 74		Self Definition 74	Definition 148 Referenz 140	TextColor Property Referenz BevelButton 161	TrimRight 138
Punkt- vor Strichrechnung 69	SelLength Property Referenz ComboBox 171		SelStart Property Referenz ComboBox 171	Steuerelemente-Hierarchie Referenz 153	TextFont Property Referenz BevelButton 161	True. Siehe Boolean Referenz 115
Purple 119	RGB 119		Serial 87, 89	Str 32	Referenz BevelButton 161	TrueWindow Referenz 154
Push Referenz 179	RGBA 119		SetBoolean Methode Referenz CheckBox 169	String 37, 128 Referenz 131	Referenz BevelButton 161	Typecast CType 84
PushButton Definition 21	Right 137		SetFocus Referenz 156	StringValue 133	Referenz BevelButton 161	Typecasting 82, 141 Definition 80
Q	RowTag Methode Referenz PopupMenu 175		Shared Computed Property 76	Structure 132	Referenz BevelButton 161	U
Querreferenz 90	Run-Event 30		Shared Property Performance 77	Sub 49 Definition 47	UBound Referenz 112	UIColor 117
R	S		Shuffle Referenz 113	Subklasse 140	UIInt8 27, 28, 125, 128	UIInt16 27, 28, 125, 128
RAD 21 Definition 16	Saturation 119			Subtraktion 34	UIInt32 27, 28, 125, 128	UIInt64 27, 28, 125, 128
RaiseEvent Definition 84	ScaleFactor 164			Super 81, 86, 140 Definition 80	UInteger 128 Definition 28	Underline Property Referenz
Rapid Application Development (System). Siehe RAD	ScaleFactorChanged Event Referenz Canvas 164			Superklasse 140		
Recent Projects 15	Schleife Definition 39			Syntaktischer Zucker Definition 73		
Rechtsklick 177	Do ... Loop 39			System.Debuglog 16		
RectControl Referenz 151	Endlosschleife 40				Then 35	
Red 119					This item does not exist 49	



BevelButton 161
CheckBox 169
PopupMenu 174
PushButton 179
unendlich. Siehe Inf
ungleich 36
Unicode 134
 Codepoint 134
 Tabelle 138
Unicode Transformation Format.
 Siehe UTF
unsigned
 Referenz 125
Unterklasse. Siehe Subklasse
Until. Siehe Do ... Loop
unveränderlich 134
Uppercase 138
UseFocusRing
 Property
 Referenz
 Canvas 165
UTF
 Definition 38
UTF-8 38, 131

V

Value 119
 Property
 Referenz
 BevelButton 161
 CheckBox 169

Variable
 Definition 27
 lokale 48
 optionale 62
 zulässige Namen 40

Variant 128, 143
 Definition 84

Vererbbarkeit 78

Vererbung 140

Vergleichsoperatoren 36

Verzweigung
 bedingte 35

Visible
 Property
 Referenz 154

Visual Basic 11

W

wchar_t* 139
wchar_t** 139
WeakRef 101
Web 15
Wend. Siehe While ... Wend
While. Siehe While ... Wend
While ... Wend
 Definition 40
White 119
Width
 Property
 Referenz 153
Window 146
 Property
 Referenz 150
WString 128
 Referenz 139

X

Xojo
 Aussprache 9
 Framework 38, 55, 134
 Geschichte 10
 Name 7
Xojo.Core.Date 55
 .Now
 Definition 55
Xojo.Core.DateInterval
 Definition 61

Y

Yellow 118, 119

Z

Zählung
 bei 0 beginnend 24
Zeichencodierung 134
Zeiger. Siehe Ptr
Zeigertyp. Siehe Datentyp: dynamisch
Zucker
 syntaktischer. Siehe Syntaktischer Zucker



Impressum

© 2016, 2017 Ulrich Bogun für Xojo, Inc.

Bis ich mich durch Open Source-Creative Commons-Lizenzen gelesen habe:

Sie dürfen diese Datei lesen, ausdrucken und unter Nennung der Quelle für eigene Schulungszwecke verwenden.

Sie sind herzlich dazu eingeladen, eigene Kapitel beizusteuern, um dies womöglich doch zum Handbuch zu machen – auch wenn es digital bleiben wird.

Xojo® ist ein eingetragenes Markenzeichen von Xojo, Inc., USA.

Alle Nennungen von Produkten und Herstellern von Produkten, ob eingetragene Markenzeichen oder nicht, dienen im Rahmen dieses Buches nur zu Schulungszwecken.

Das Xojo-Handbuch wird privat zum privaten Download angeboten.

Es ist nicht im Verzeichnis des Buchhandels eingetragen und trägt keine ISBN.

Auch wenn der Autor sich um größtmögliche Sorgfalt bemüht hat: Es wird keine Garantie für die Funktionsfähigkeit der Beispielprogramme übernommen, insbesondere im Hinblick auf mögliche zukünftige Xojo-Erweiterungen. Sie verwenden eventuell beiliegende Beispielprojekte oder selbst aus dem Nichthandbuch übertragene Programme auf eigenes Risiko. Eine Haftung für aus Fehlbedienung oder Fehlfunktion erfolgenden Konsequenzen ist ausgeschlossen. Allerdings: Sie müssten schon sehr, sehr, sehr weit von den Beispielen abweichen, um Ihre Daten auch nur potenziell gefährden zu können!

Preis: 0 €.

Ähnlich wie bei meinen freien Xojo-Bibliotheken gilt auch hier:

Ich habe gar nicht mal so wenig Zeit reinvestiert. Ich bin durch meine Beschäftigung als Xojo-Entwicklerevangelist in einer etwas privilegierten Lage und darf beruflich mit diesem Entwicklungssystem spielen. Davon gebe ich gerne zurück – so eben wie in dieser Form – in der Hoffnung, es möge helfen, Xojo hierzulande zum Stellenwert zu verhelfen, den die IDE als unschlagbar einfach zu begreifendes und doch funktionsmächtiges Programmiererwerk verdient hat.

In der Praxis, so viel mag ich verraten, überschreitet die investierte Zeit so manches Mal den verträglichen Rahmen. Neben einer Lobpreisung auf die Leidensfähigkeit meiner Lebensgefährtin: Verständlich, dass ich weitere Arbeiten immer wieder von meiner allgemeinen Auftragslage abhängig machen muss. Sollten Sie sich Ihrerseits revanchieren wollen und weitere Kapitel voranbringen, fühlen Sie sich frei, Paypal anzuwerfen und eine Autorenspende an bogun@zeichen-satzservice.de zu senden. Gleiche Adresse nimmt auch gerne Ihre Kritik oder Anregungen entgegen.

Vielen Dank!