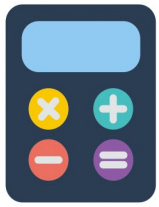
Hauke Fehr











EIGENE APPS PROGRAMMIEREN

Ohne Vorkenntnisse einsteigen

- Programmieren lernen mit LiveCode
- Apps erstellen für Windows, Android und macOS
- Spaß mit Multimedia, Spielen, Sprachausgabe u. v. m.

Alle Beispielprojekte aus dem Buch zum Download

Rheinwerk Computing

Kapitel 2
Das App-Werkzeug: LiveCode

LiveCode ist die Umgebung, in der Sie sich bald ganz zu Hause fühlen werden. Hier erstellen Sie Ihre Projekte, zunächst ganz einfach per Klick und Drag-and-drop, dann machen Sie sie mit eigenen Skripten lebendig. Später erstellen Sie aus ihnen Apps für Ihre bevorzugte Plattform.

LiveCode ist im Vergleich zu vielen anderen großen Entwicklungsumgebungen, die mit gigabytegroßen Paketen und zahlreichen Betriebssystemeingriffen stundenlange Installationssessions durchziehen, ein ganz kleines Programm, das mal eben schnell und einfach installiert werden kann. Das sollten Sie als Erstes auch tun! In den Materialien zum Buch finden Sie die stabile Version 9.5 der Community Edition von LiveCode. Wir empfehlen, dass Sie diese Version installieren, um die Beispiele in diesem Buch nachzubauen.

LiveCode befindet sich in stetiger Entwicklung. Es ist also gut möglich, dass zu dem Zeitpunkt, zu dem Sie dieses Buch lesen, bereits eine neuere Version von LiveCode erhältlich ist – das ist aber nicht weiter schlimm, denn Version 9.5 macht all das, was wir hier machen wollen, stabil, fehlerfrei und genau wie beschrieben. Außerdem beziehen sich alle Beispiele und Anleitungen auf diese Fassung. Wenn Sie jedoch gerne die aktuellste Version laden und ausprobieren wollen, können Sie das jederzeit hier tun:

http://downloads.livecode.com/livecode/

Hier finden Sie absolut alle bislang erschienenen Versionen von LiveCode. Die kostenlose (OpenSource-Version) ist immer die *Community Edition*, und die Abkürzungen davor besagen, wie stabil oder experimentell die Version ist. Normalerweise sollten Sie für die Entwicklung und das ernsthafte Arbeiten immer eine Version wählen, die den Zusatz *STABLE* trägt, das heißt stabil. RC bedeutet *Release Candidate*, also kurz vor Erscheinen bzw. fast ganz stabil. Damit kann man in der Regel auch schon gut arbeiten, und manchmal sind in diesen Versionen erweiterte und neue Funktionen enthalten bzw. Anpassungen für die neuesten mobilen Systeme. DP steht für *Developer Preview* – »Entwickler-Vorschau-Version«. Diese ist vor allem für Entwickler gedacht, die testen wollen, inwieweit neue Funktionen von LiveCode bereits funktionieren. Für die Entwicklung endgültiger Apps empfehlen wir sie nicht. Dafür sind die stabileren Fassungen gedacht.

Was bedeutet Community Edition?

Die Community Edition von LiveCode ist Open Source und kostenlos. Sie enthält alle Elemente von LiveCode, und man kann damit auch technisch alles machen. Was man damit nicht machen darf, ist das Erstellen, Verbreiten und Verkaufen kommerzieller Programme. Die Lizenz erlaubt nur die Verbreitung kostenloser Programme, die keinem kommerziellen Zweck dienen, und erfordert auch das Offenlegen, das heißt Bereitstellen des Quellcodes des Programms für jeden Interessierten. Zudem erlaubt Apple nicht die Verbreitung von Open-Source-Software im AppStore. Das bedeutet: Natürlich können Sie mit der Community Edition entwickeln, was immer Sie wollen, und es auf Ihrem Gerät testen. Und wenn ein Programm so gut wird, dass Sie es gern über Apples Store verbreiten möchten, dann müssen Sie noch die Indy-Lizenz für LiveCode erwerben. Für eine entsprechende Jahresgebühr können Sie die mit LiveCode erstellten Apps dann auch auf allen Plattformen vermarkten und zu Geld machen. Aber immer eines nach dem anderen!

2.1 Installation unter Windows

Starten Sie die Datei *LiveCode_Community_Installer_Windows.exe* von den Materialien zum Buch, und es öffnet sich das in Abbildung 2.1 dargestellte Fenster.

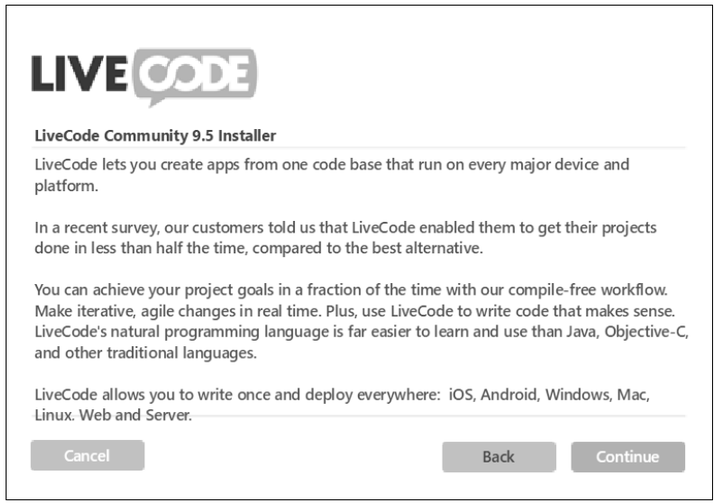


Abbildung 2.1 Der Windows-Installer

Nach dem Klick auf CONTINUE muss noch das Open-Source-Lizenzabkommen bestätigt werden (Haken klicken), dann geht es weiter.

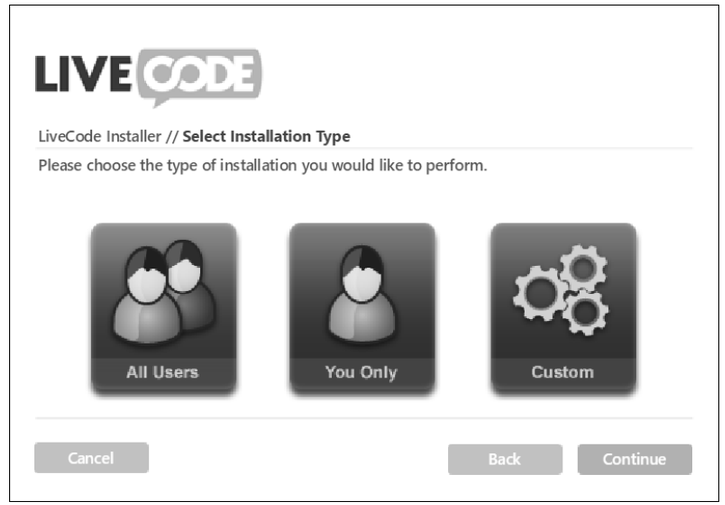


Abbildung 2.2 Installationstyp auswählen

Im nächsten Bildschirm können Sie ALL USERS oder YOU ONLY wählen, je nachdem, ob das Programm von mehreren angemeldeten Benutzern in Windows verwendet werden soll oder nicht (Abbildung 2.2). Ich empfehle YOU ONLY.

Im nächsten Fenster klicken Sie einfach INSTALL – und nach wenigen Sekunden haben Sie LiveCode auf Ihrem PC installiert. Ja, das war's schon. Nicht einmal ein Neustart ist erforderlich. Auf dem Desktop finden Sie das Icon, mit dem Sie LiveCode starten können, und Sie sind komplett bereit, loszulegen!

2.2 Installation auf dem Mac

Ähnlich einfach wie unter Windows lässt sich LiveCode auch auf dem Mac installieren. Öffnen Sie die Image-Datei *LiveCode_Community _9_5_Mac.dmg* aus den Materialien zum Buch durch Doppelklick. Darin finden Sie bereits die LiveCode-App, die Sie einfach nur nach unten in den dort angezeigten Ordner *Applications* (bzw. *Anwendungen*) ziehen müssen (Abbildung 2.3).

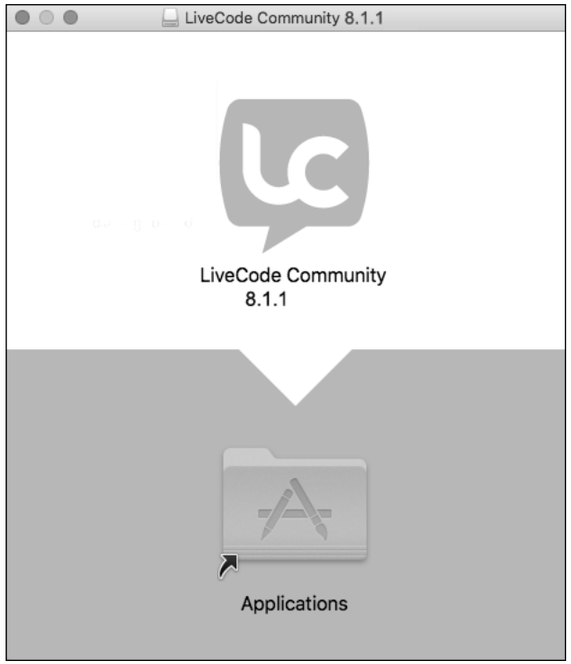


Abbildung 2.3 Installationsprozess auf dem Mac – einfacher geht es nicht!

Das System kopiert jetzt die Dateistruktur, was ein paar Sekunden, vielleicht auch eine Minute dauern kann. Danach befindet die App sich dort, wo sie hingehört – nämlich im Ordner *Anwendungen*, und von dort können Sie sich eine Verknüpfung in Ihr Dock ziehen, damit Sie LiveCode jederzeit schnell und einfach starten können.

Beim ersten Start müssen Sie dem Mac eventuell noch erlauben, LiveCode auszuführen, danach geht es dann immer sofort. Auch beim Mac gilt: Nach wenigen Minuten sind Sie startklar und bereit für das Erstellen Ihrer eigenen Apps. Jetzt kann es also losgehen!

> Wie kommen meine Apps dann auf Handy und Tablet?

Nur damit keine Missverständnisse entstehen: Unsere Entwicklungsumgebung befindet sich immer auf dem Mac oder PC. Wir erstellen und programmieren unsere Apps grundsätzlich als Projekte auf dem klassischen Computer mit Maus und Tastatur. Jedes unserer LiveCode-Projekte wird zunächst hier erstellt.

Wenn das Projekt dann fertig ist, können wir daraus anschließend verschiedene Arten von Apps machen. Wir können zum Beispiel Apps für Windows oder Mac erstellen. Das geht am schnellsten und einfachsten. Anschließend verhält sich unsere fertige App ganz genau so wie das Projekt in LiveCode, das wir erstellt haben, nur dass es kein LiveCode mehr braucht, sondern ein »richtiges Programm« wie alle anderen Programme auf unserem Computer ist und wir diese App dann an unsere Freunde, an die Community oder unsere Kunden weitergeben können.

Die App soll jetzt aber auf das Handy oder Tablet? Auch diese Apps erstellen Sie erst einmal auf Ihrem Computer. Sie achten nur von Anfang an darauf, dass sich das Bildformat für das mobile Gerät eignet (hochkant oder quer, passendes Breiten-/Höhenverhältnis usw.) und dass die Bedienelemente groß genug und geeignet für die Touch-Bedienung sind. Wenn das LiveCode-Projekt am Computer fertiggestellt wurde und funktioniert, können Sie es in eine APK-Datei (für Android) umwandeln lassen. Das sind die Dateiformate einer lauffähigen App für Android-Geräte wie Mobiltelefone oder Tablets. Das macht LiveCode für Sie – vorausgesetzt, Sie haben Ihr System entsprechend konfiguriert und zusätzliche Werkzeuge von Google eingerichtet. Diese Datei kann dann anschließend im Smartphone- oder Tablet-Simulator ausprobiert werden, sie kann zum Testen auf dem eigenen mobilen Gerät installiert werden bzw. später sogar unter bestimmten Bedingungen in den Google Play Store eingereicht werden. Wie das genau geht und was man dabei alles beachten muss, wird in Kapitel 17, »Apps auf Handys und Tablets bringen«, und aktualisiert im Webanhang des Buches erklärt und besprochen. Prinzipiell können Sie Ihre Apps mit LiveCode auch auf iOS-Geräte wie iPhone oder iPad bringen. LiveCode erstellt auch dafür die benötigten Dateien. Aufgrund der sehr restriktiven Politik von Apple ist es allerdings rechtlich nicht möglich, OpenSource-Programme für iOS-Geräte anzubieten. Das heißt, um iOS-Apps zu erstellen, benötigen Sie eine kommerzielle (und kostenpflichtige) Version von LiveCode sowie zusätzlich einen kostenpflichtigen Apple Developer Account. Für Profis, die ihre Apps verkaufen und auf alle Plattformen bringen wollen, lohnt sich das gewiss. Zum Ausprobieren ist es jedoch ratsamer, sich erst einmal mobil auf Android zu beschränken. Wir werden uns daher vorrangig mit Android beschäftigen, da dieses Buch die Möglichkeiten mit der kostenlosen Fassung von LiveCode beschreibt – wenn Sie kommerziell für iOS entwickeln wollen, finden Sie natürlich alle speziellen Informationen hierzu auf den Webseiten von LiveCode.

Dieses Buch handelt in erster Linie davon, wie Sie mit Spaß und kreativen Ideen selber Apps programmieren lernen und in kurzer Zeit schon Ihre eigenen Vorstellungen in funktionierende Programme umsetzen können. In den folgenden Kapiteln werden wir uns intensiv damit beschäftigen, wie man LiveCode-Projekte erstellt und wie man

grundlegende und durchdachte Programme baut, die dann später prinzipiell als App auf jeder Plattform laufen können.

2.3 Erster Start – es geht sofort los

Genug vorbereitet. Jetzt soll es endlich losgehen! Starten Sie LiveCode mit dem grünen Lc-Icon, und nach wenigen Sekunden sind Sie »drin« (die Aufforderung, sich zu registrieren, können Sie gerne mit SKIP THIS STEP überspringen).

> **Abbildungen zeigen Windows-Version von LiveCode**
>
> Nahezu alle Abbildungen in diesem Buch zeigen die Elemente der Windows-Version von LiveCode. Funktional sind sie aber völlig identisch mit der Mac-Version. Außer dass die Fenster am oberen Rand etwas anders aussehen und die Buttons und Felder verschiedene Ränder und Rundungen haben, verhalten sie sich genau gleich. Das ist übrigens auch das Praktische an LiveCode: Ein Projekt, das in Windows gebaut wurde, kann ebenso auf dem Mac geöffnet werden und funktioniert auch da sofort, nur dass manche Elemente ein klein wenig anders aussehen – angepasst an das jeweilige Betriebssystem.

Das Start-Center, das sich möglicherweise öffnet, können Sie gern erst einmal schließen. Vielleicht haben Sie später Lust, ein paar der vorgeschlagenen Hilfsfunktionen auszuprobieren, aber jetzt verzichten wir erst einmal darauf. Sie können es mit der Checkbox rechts unten auch permanent abstellen, denn an die Funktionen kommen Sie später auch auf anderen Wegen.

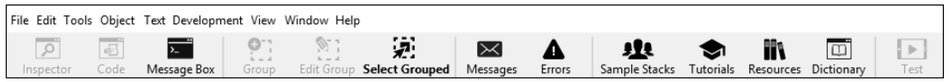


Abbildung 2.4 Die Hauptleiste des LiveCode-Systems

Im Hauptprogrammfenster der LiveCode-Entwicklungsumgebung finden Sie das Menü mit allen wichtigen Funktionen des Entwicklungssystems sowie darunter zahlreiche Icons, mit denen sich häufig benutzte Elemente direkt aufrufen lassen (Abbildung 2.4). Beim Mac ist das Menü wie üblich ganz oben am Bildschirmrand ausgelagert, und die Leiste enthält nur die Symbole.

Das zweite sehr wichtige Element ist die senkrechte *Werkzeugpalette* (TOOLS PALETTE), die immer automatisch mit erscheint und stets verfügbar sein sollte. Falls diese aus irgendeinem Grund nicht zu sehen ist, wählen Sie bitte aus dem Menü der Hauptleiste TOOLS • TOOLS PALETTE, dann erscheint sie auf jeden Fall.

Abbildung 2.5 Die Werkzeugpalette in LiveCode 9.5

In der Werkzeugpalette finden Sie alle Elemente, die Sie später in Ihr Projekt ziehen können und in Ihrer App zum Leben erwecken. Hier gibt es Buttons, Textfelder, Listenelemente, Dialoge, Bedienknöpfe, Grafik- und Bildfelder, Player, Widgets und vieles mehr, das Sie noch ausgiebig verwenden werden.

Wo aber spielt sich nun das eigentliche Projekt, die App, ab? Noch gar nicht, denn Sie haben ja noch gar kein Projekt gestartet. Das sollten Sie jetzt also machen. Wählen Sie in der oberen Hauptleiste den ersten Menüpunkt FILE, dort den ersten Eintrag NEW STACK und dann rechts DEFAULT SIZE. Warum es »New Stack« heißt und was das alles soll, werde ich später noch genau erläutern. Erst einmal reicht es, wenn Sie es genauso machen wie beschrieben.

Es öffnet sich ein leeres Fenster mit der Beschriftung UNTITLED 1 * (Abbildung 2.6). Das ist Ihr Projekt, die Basis für Ihre erste App!

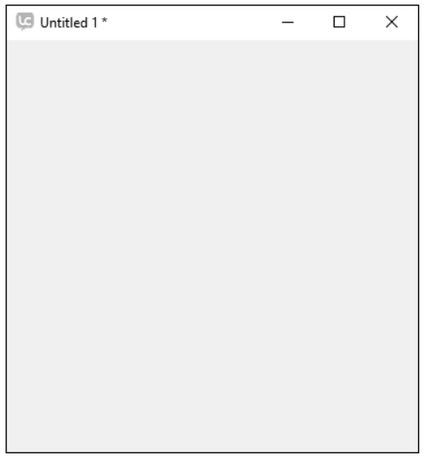


Abbildung 2.6 Das erste Projekt – noch völlig leer, aber voller Möglichkeiten!

Großartig! Das erste Projekt ist da, quasi die erste App. Ein echtes Programmfenster. Das ist doch mal ein Anfang!

Grundwerkzeuge und Menüs

Bevor Sie beginnen, damit herumzuspielen, Elemente hinzuzufügen und dieses erste Projekt ganz nach eigenem Geschmack zu gestalten, müssen Sie eine ganz wichtige Sache wissen: In LiveCode gibt es zwei Modi, den *Bearbeitungsmodus* und den *Laufzeitmodus*. Im Bearbeitungsmodus können Sie Elemente auf dem Programmfenster auswählen, platzieren, hin und her bewegen, vergrößern, verkleinern und Attribute aller Art setzen. Es ist eine Art Werkzeugkastenmodus, in dem Sie Ihre App zusammenklicken und gestalten. Wechseln Sie in den Laufzeitmodus, dann können Sie die Elemente nicht mehr herumschieben oder verändern, sondern dann *läuft die App bereits*, und Sie können sie testen, die Buttons klicken, die Textfelder ausfüllen usw.

> Wie schaltet man zwischen den Modi um?

Ganz oben in der Werkzeugpalette gibt es zwei nebeneinanderliegende Mauspfeile (Abbildung 2.7). Der linke ist ein normaler Pfeilcursor, der rechte ist ein Cursor mit Größenkreuz.

Abbildung 2.7 Die beiden Bearbeitungsmodi werden hier ausgewählt.

Klicken Sie auf den linken Cursor (der auch BROWSE TOOL heißt), dann sind Sie im Laufzeitmodus und können Ihr Programm im realen Betrieb testen. Hier verhalten sich alle Elemente so, wie Sie sie programmiert haben, es werden Buttons geklickt und Skripte ausgeführt – das Programm »läuft«. Klicken Sie auf den rechten Pfeil (POINTER TOOL), dann kommen Sie in den Bearbeitungsmodus. In diesem Modus werden Sie immer beginnen, wenn Sie die Elemente Ihres Projekts zusammenstellen. Hier legen Sie Elemente an, gestalten sie, schieben sie hin und her, ändern ihre Größe, Form und Farbe usw., legen die Skripte an. Später werden Sie ständig zwischen den beiden Modi hin- und herwechseln, bauen – testen – bauen – testen. Jetzt probieren wir es einfach mal aus!

Elemente auf das Hauptfenster ziehen

Achten Sie darauf, dass Sie sich im Bearbeitungsmodus befinden. Wenn nicht, dann klicken Sie das POINTER-TOOL, sodass es grau unterlegt ist.

Nun gehen Sie mit der Maus auf eines der Button-Symbole auf der Werkzeugpalette, drücken die linke Maustaste und ziehen das Symbol mit gedrückter Maustaste in Ihr leeres Programmfenster. Dort lassen Sie die Maustaste los.

Weil Sie im Bearbeitungsmodus sind, erscheint der Button jetzt bearbeitbar, ist angewählt und hat Anfasser an den Ecken und Kanten (Abbildung 2.8). Sie können mit der Maus nun seine Größe ändern, indem Sie auf die Anfasser gehen, die linke Maustaste drücken und mit gehaltener Maustaste die Ecken und Kanten verschieben. Wenn Sie mit dem Mauscursor in die Mitte des Buttons gehen und die linke Maustaste drücken und halten, können Sie den Button beliebig innerhalb des Programmfensters verschieben. Probieren Sie das alles einmal ausgiebig aus!

Jetzt probieren Sie einmal aus, was passiert, wenn Sie in den Laufzeitmodus wechseln. Klicken Sie mit der Maus in der TOOL PALETTE ganz oben auf das Pfeilsymbol BROWSE-TOOL, sodass es grau unterlegt ist. Anschließend verhält sich Ihr Programmfenster schon wie ein richtiges eigenes Programm. Es gibt einen Button, der aktiv wird, wenn man mit der Maus darauf zeigt. Wenn man ihn klickt, lässt er sich herunterdrücken und kommt wieder hoch, wenn man ihn loslässt. Ein echtes Programmfenster mit einem lebendigen Button. Wenn man so will, ist das schon Ihr erstes Programm, Ihre erste App!

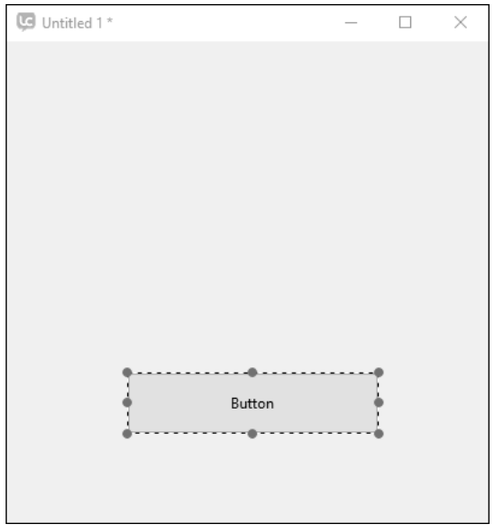


Abbildung 2.8 Button-Größe und -Position lassen sich beliebig anpassen.

Klar, es passiert ansonsten nichts, wenn Sie den Button klicken. Kann ja auch nicht, denn Sie haben dem Programm noch keinerlei Anweisungen gegeben, was es machen soll. Also macht es auch nichts weiter. Alles ist, wie es sein soll. Mit Leben füllen wir das Programm später.

Probieren geht über studieren

Nun steht es Ihnen erst einmal völlig frei zu probieren, was es da noch alles für schöne Elemente gibt, die Sie in Ihr Programmfenster ziehen können. Zurück in den Bearbeitungsmodus!

Es gibt in der Werkzeugpalette mehrere Sektionen. Die oberste enthält den Umschalter zwischen den beiden Modi. Darunter folgt die *Widget*-Sektion. Das sind ziemlich komplexe Elemente wie eine Uhr, ein Webbrowser, verschiedene gestylte Leisten, Schalter, Anzeiger für mobile Geräte usw. Widgets sind wie kleine Programmmodule in sich, die schon eine Menge können, ohne dass man sie programmieren muss. Erfahrene Programmierer können selber Widgets schreiben und hinzufügen. Sie brauchen sich aber darum jetzt keine Gedanken zu machen, sondern können einfach mal ausprobieren, wie sie auf Ihrem Programmfenster aussehen, indem Sie sie hineinziehen.

Unter den Widgets kommen die *Standardobjekte*, die man in vielen normalen Apps benötigt, wie Buttons, Textfelder, Tabellen, Checkboxen, Auswahlelemente, Anzeiger, Player usw. Ziehen Sie sie in Ihr Programmfenster, und schauen Sie mal, wie sie aussehen und sich dort einfügen.

Darunter finden Sie auch noch Grafiken und Grafikgestaltungselemente. Experimentieren Sie auch damit herum! Später werden Sie lernen, wo Sie sie bei Bedarf sinnvoll einsetzen können. Jetzt geht es erst mal ans »Draggen and Droppen«! Klicken Sie sich ein schönes Programmfenster zusammen! Testen Sie möglichst viele Elemente, ziehen Sie Grafiken auf (bei Linien und Polygonen beenden Sie die Erstellung per Doppelklick). Verändern Sie Größen und Positionen, schalten Sie um auf den Laufzeitmodus, und probieren Sie die Elemente aus: Schalten Sie die Schalter, schieben Sie die Schieber, schreiben Sie Texte in die Textfelder (Abbildung 2.9).

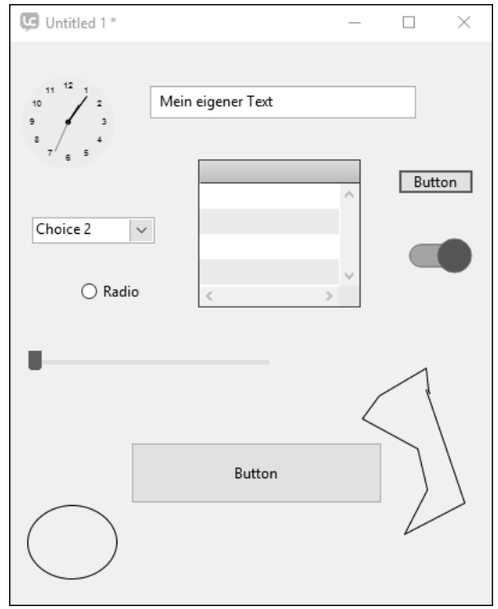


Abbildung 2.9 Im Nu kann man eine Menge Objekte ins Programmfenster ziehen und auch gleich bedienen.

Toll, was man da in kürzester Zeit zustande bekommt. Und alles ist gleich voll »benutzbar«, wenn auch noch nichts weiter passiert.

Schriften, Farben und sonstige Eigenschaften

Was Sie da jetzt auf dem Programmfenster platziert haben, sind *Objekte*. Das heißt, es sind Elemente Ihres Programms, die bestimmte Eigenschaften haben (wie zum Beispiel Größe und Position) und die auf Eingaben und Ausgaben reagieren können – womit Sie sich bald mehr beschäftigen werden.

Jetzt schauen wir uns noch einmal genauer die vielen Eigenschaften der einzelnen Objekte an. Es gibt natürlich nicht nur *Größe* und *Position*, die wir mit der Maus verändern können. Jede Art von Objekt kann ganz unterschiedliche weitere Typen von Eigenschaften haben.

Gehen Sie in den Bearbeitungsmodus, und wählen Sie mit der Maus einen der Buttons an, die Sie erstellt haben. Wenn der Rahmen mit den Anfassern erscheint, ist der Button ausgewählt. Jetzt schauen Sie sich eine Auflistung aller Eigenschaften dieses Buttons an. Das machen Sie mit dem sogenannten *Property Inspector*. Properties sind Eigenschaften eines Objekts, und mit dem Property Inspector können Sie die Eigenschaften jedes Objekts betrachten und auch verändern.

Property Inspector

Es gibt drei Wege, um den Property Inspector für ein ausgewähltes Objekt aufzurufen:

- Klicken Sie oben im Hauptmenüfenster auf das erste Symbol INSPECTOR.
- Klicken Sie mit der rechten Maustaste (Ctrl + Mausklick beim Mac) auf das ausgewählte Objekt, und wählen Sie im Kontextmenü PROPERTY INSPECTOR aus.
- Führen Sie einen Doppelklick mit der Maus auf das ausgewählte Objekt aus.

Egal, wie Sie es machen, am Ende geht auf jeden Fall immer der PROPERTY INSPECTOR auf und sieht bei Ihrem Button etwa so aus wie in Abbildung 2.10.

Der Inspector zeigt Ihnen eine Menge Informationen über den Button an. Er öffnet sich mit der Hauptseite, hat aber noch acht weitere Seiten, die sich über die kleinen Symbole oben im Inspector umschalten lassen.

Zu den wichtigsten Eigenschaften (*Properties*) des Buttons gehören zum einen sein *Objektname*, zum anderen seine Aufschrift (*Label*). Der Name ist wichtig, damit Sie den Button später eindeutig identifizieren und ihn mit Ihrem Skript auch ansprechen können. Jedes Objekt sollte stets einen eindeutigen Namen erhalten.

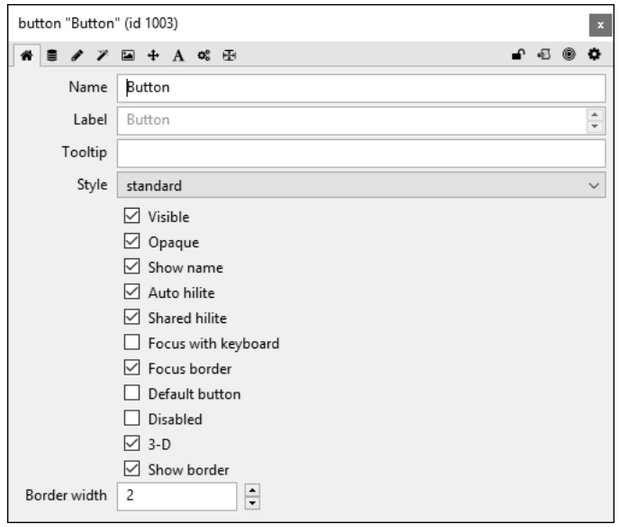


Abbildung 2.10 Der Eigenschafteninspektor (Property Inspector) für einen Button

Jedem Objekt seinen Namen

Es ist prinzipiell möglich, unterschiedlichen Objekten denselben Namen zu geben, da LiveCode die Objekte intern immer nur anhand ihrer ID unterscheidet. Sie sollten das aber unbedingt vermeiden und sich angewöhnen, die Objekte immer eindeutig zu benennen und mit ihrem Namen zu verwenden.

Schreiben Sie also zum Beispiel »meinButton« in die oberste Namenszeile. Am besten – das ist meine Empfehlung, um sauber zu arbeiten – vergeben Sie Namen immer ohne Leerzeichen. In die nächste Zeile schreiben Sie dann zum Beispiel »Klick mich!« (Abbildung 2.11). Das wird die Aufschrift des Buttons, sein LABEL.

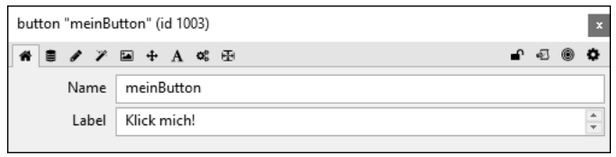


Abbildung 2.11 Änderung des Namens und der Aufschrift des Buttons

Sobald Sie mit dem Cursor in eine andere Zeile wechseln, ist die Änderung des Buttons bereits im Programmfenster zu sehen (Abbildung 2.12).

Abbildung 2.12 Button mit eigener Aufschrift

Jetzt haben Sie einen Button mit Ihrer individuellen Aufschrift (Sie können selbstverständlich auch etwas anderes als »Klick mich!« hineinschreiben). Es gibt aber noch weit mehr Eigenschaften des Buttons, die Sie selbst verändern können. Werfen Sie noch einmal einen Blick auf die erste Seite des PROPERTY INSPECTOR. Achten Sie dabei immer darauf, dass der gewünschte Button wirklich ausgewählt ist und Sie im Bearbeitungsmodus sind.

Die dritte Textzeile im Inspector heißt TOOLTIP. Wenn Sie dort etwas hineinschreiben, erscheint dies beim Überfahren des Buttons mit der Maus in einem kleinen gelben Fenster (also eine eingebaute Erklärung für den Button, wie wir sie in vielen Programmen vorfinden).

Danach kommen zahlreiche Checkboxen, die man an- oder abwählen kann. Wählen Sie zum Beispiel die Eigenschaft VISIBLE (sichtbar) ab, dann ist der Button nicht mehr auf dem Programmfenster zu sehen. Er ist noch da, aber ausgeblendet. Das ist jetzt nicht sinnvoll, aber es kann später in Ihrer App durchaus Sinn machen, wenn ein Button nur unter bestimmten Bedingungen erscheint und sonst ausgeblendet ist. Schalten Sie den Button bitte anschließend gleich wieder sichtbar, denn sonst können Sie ihn später nicht mehr anwählen (das ginge dann nur noch mit dem PROJECT BROWSER, über den Sie bald auch mehr erfahren werden). OPAQUE bedeutet undurchsichtig – hiermit wählen Sie, ob der Button eine solide oder eine durchsichtige Oberfläche hat. Die Eigenschaft DISABLED (deaktiviert) legt fest, ob der Button überhaupt aktiv ist oder nicht klickbar.

Die weiteren Eigenschaften auf der Hauptseite des Buttons wirken sich auf den Rand des Buttons und seine Eigenschaften beim Fokussieren aus. Sie können sie gern schon einmal ausprobieren, auch wenn nicht immer sofort sichtbar ist, was da passiert – diese Eigenschaften werden teilweise später noch aktuell und genauer erläutert.

Der PROPERTY INSPECTOR hat, wie gesagt, noch weitere Seiten, und als Nächstes schauen wir auf die Seite *Colors* (Farben). Sie gelangen auf sie, indem Sie im PROPERTY INSPECTOR das dritte Symbol von links, den Stift, klicken.

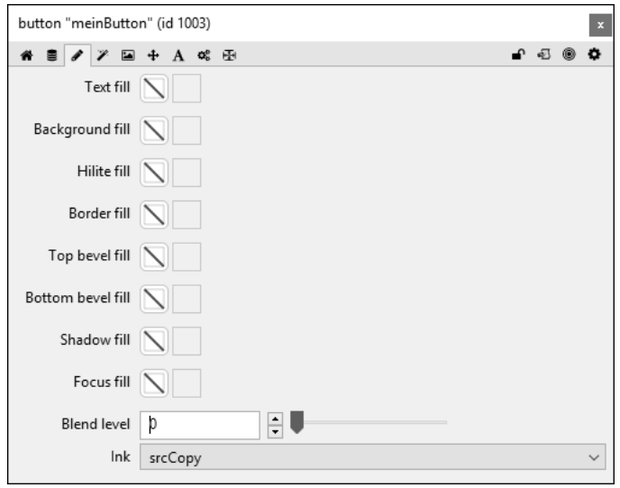


Abbildung 2.13 Farben und Muster – Eigenschaften des Buttons

In der ersten Zeile TEXT FILL können Sie die Textfarbe des Buttons einstellen (Abbildung 2.13). Klicken Sie dazu auf das linke Quadrat mit dem roten diagonalen Balken (das bedeutet »keine Farbe gewählt«). Im Farbwähler, der sich nun öffnet, können Sie jede beliebige Farbe für den Text auswählen. Nach Klick auf OK können Sie die geänderte Textfarbe sofort im Button auf dem Programmfenster sehen.

Mit der Eigenschaft BACKGROUND FILL können Sie dasselbe für die Hintergrundfarbe, also die Hauptfläche des Buttons, machen. Damit können Sie dem Button ein ganz individuelles farbliches Aussehen geben.

Mit dem Regler BLEND LEVEL ganz unten ist es übrigens möglich, den Button mehr oder weniger durchscheinend zu machen. Wer weiß – vielleicht können Sie das ja mal gebrauchen.

Die anderen Properties im Farbmenü können Sie gern ausprobieren. Sie setzen Farben für Details im Rahmen, Schatten oder Auswahlrahmen des Buttons. Statt Farben lassen sich mit dem rechten Viereck neben jeder Eigenschaft auch Muster als Füllung verwenden. Kann man machen, muss man nicht.

Die nächste Seite im PROPERTY INSPECTOR, erreichbar über das Zauberstabsymbol, heißt GRAPHICS EFFECTS (Grafikeffekte), und wie der Name schon sagt, können Sie damit das gewünschte Objekt grafisch ein wenig aufpeppen. Es lassen sich Schlagschatten setzen, äußeres und inneres Glühen einstellen und eine Überlagerungsfarbe wählen (Abbildung 2.14). Wer kreativ ist und fleißig herumexperimentiert, kann das Aussehen

eines Objekts damit schon richtig professionell gestalten und braucht möglicherweise gar keine externen Grafiken mehr. Später mehr dazu!

Mit der Seite ICONS, erreichbar über das Bildsymbol in der Menüleiste, lassen sich beliebige Grafiken als Bilder auf dem Button platzieren, auch verschiedene für unterschiedliche Zustände (inaktiv, aktiv, gedrückt). Es können auch Bilder den Button vollständig ersetzen. Dazu ebenfalls später mehr. Aber Ausprobieren ist natürlich jederzeit erlaubt!

Abbildung 2.14 Button mit Grafikeffekten und Icon

Wichtig sind jetzt erst einmal noch zwei Property-Seiten, zunächst die Seite POSITION, in der Sie die Größen- und Positionsdaten des Buttons in exakten Zahlen erhalten und entsprechend auch anpassen können (Abbildung 2.15).

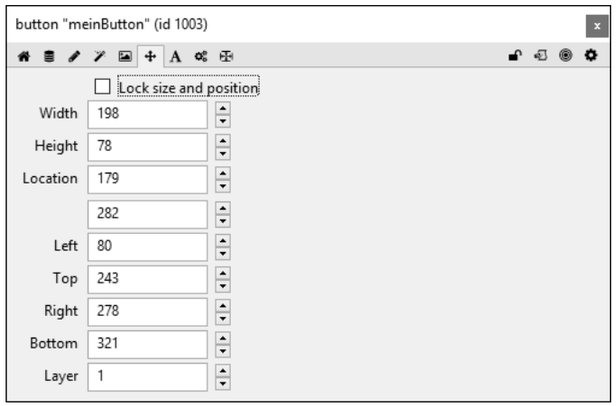


Abbildung 2.15 Weitere Eigenschaften des Buttons: Maße und Position

Neben den Werten für Breite und Höhe sowie dem Abstand von linkem, rechtem, oberem und unterem Rand des Programmfensters können Sie hier auch den LAYER für den Button festlegen. Das ist gewissermaßen die Ebene für die Darstellung des Objekts. Wenn es mehrere Buttons oder andere Objekte gibt, die übereinanderliegen, dann liegt das mit der höheren Layer-Zahl oben.

Zu guter Letzt schauen wir uns noch die Seite für die Texteigenschaften des Buttons an, die Seite TEXT.

Abbildung 2.16 Texteigenschaften – Größe und Schriftart

Hier können Sie die Schriftart (FONT) und die Schriftgröße für den Button wählen (Abbildung 2.16), außerdem den Textstil wie Fett oder Kursiv. Nur die Textfarbe wird hier nicht gesetzt – das geschieht ja, wie schon gesehen, auf der Seite COLOR.

Anhand des Buttons haben Sie jetzt eine Menge Möglichkeiten ausprobiert, die Eigenschaften eines Objekts nach Belieben zu verändern. Das geht natürlich nicht nur mit Buttons, sondern mit allen Objekten, sogar mit dem Programmfenster (dem STACK) selbst. Wenn Sie bei offenem PROPERTY INSPECTOR auf eine leere Stelle des Programmfensters klicken, dann sehen Sie den PROPERTY INSPECTOR für den STACK und können hier auch Hintergrundfarben, Fenstertitel und vieles mehr einstellen (Abbildung 2.17).

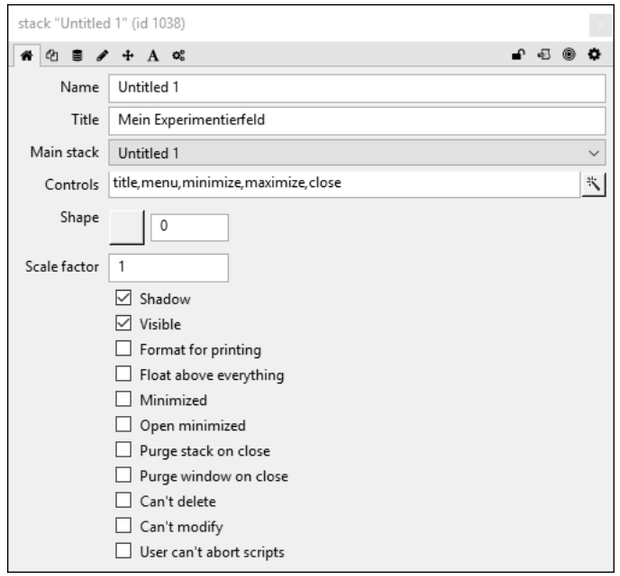


Abbildung 2.17 Eigenschaften des Programmfensters (Stack)

Wenn Sie hier »Mein Experimentierfeld« oder irgendetwas anderes in die Zeile TITLE schreiben, zeigt das Hauptprogrammfenster automatisch diesen Namen in der Titelzeile. Der Name des Stacks ist der Name Ihres Programms.

Übrigens: Eigenschaften wie Schriftgrößen und Hintergrundfarben, die Sie hier im Programmfenster vornehmen, wirken sich automatisch auf alle Objekte aus, die sich in diesem Fenster befinden – außer es sind für sie explizit andere Werte gesetzt. Warum das so ist, wird noch genauer erläutert. Es hat mit der Hierarchie der Objekte zu tun.

Jetzt ist die Manege wieder frei für Sie: Probieren Sie, so viele Eigenschaften wie möglich in Ihren Objekten zu verändern. Machen Sie Textfelder farbig, schreiben Sie einen anderen Text rein, machen Sie die Rahmen fett, geben Sie den Grafiken Schatten, gestalten Sie ein chaotisches oder ein geordnetes Fenster mit möglichst vielen Elementen. Das gibt Ihnen ein Gefühl dafür, wie Sie Ihre eigene App später gestalten können und was es für Möglichkeiten gibt. Wenn noch nicht alles klappt, was Sie sich vorstellen, dann ist das nicht weiter schlimm. Anhand der Beispiele, die in diesem Buch noch folgen, werden Sie eine Menge nützliche Rezepte erhalten, wie Sie ganz individuelle Ideen präzise umsetzen können. Alles kommt nach und nach – jetzt ist erst mal Freestyle angesagt!

Wenn Sie zufrieden sind und das Gefühl haben, ein wenig Kontrolle darüber zu besitzen, wie man die Objekte auf dem Programmfenster anordnen und schön anpassen kann, können Sie ja auch schon einfach mal Ihre erste App erstellen. Jawohl, eine App ohne Programmierung, nur durch Zusammenklicken und Setzen von Eigenschaften! Sie ist in wenigen Minuten erstellt.

Weltzeituhr mit Webnachrichten: Klicken Sie sich eine App zusammen!

Beginnen Sie mit einem ganz frischen Projekt. Wenn Sie noch das Fenster und die Objekte vom Ausprobieren vor sich haben, dann wählen Sie aus dem Hauptmenü von LiveCode FILE • CLOSE AND REMOVE FROM MEMORY. Damit schließen Sie das aktuelle Projekt und entfernen es aus dem Speicher. Oder was auch immer klappt: Beenden Sie LiveCode, indem Sie das Hauptmenüfenster schließen, und starten Sie es neu. Meist geht auch das in wenigen Sekunden. Nun starten Sie ein neues Projekt über FILE • NEWS STACK • IPHONE 6, um eine passende Größe für eine typische Smartphone-App zu haben (das passt auch sehr gut für die meisten Android-Phones).

Es kann losgehen: Ziehen Sie zwei Uhr-Widgets aus der Werkzeugpalette auf das Programmfenster. Platzieren Sie sie nebeneinander in der oberen Hälfte des Fensters. Vergrößern Sie sie etwas, und zwar 100 × 100 Pixel groß. Dazu wählen Sie die erste Uhr aus (Bearbeitungsmodus), und wechseln Sie in den PROPERTY INSPECTOR. Hier wählen Sie

die Seite POSITION, stellen als WIDTH (Breite) »100« ein und als HEIGHT (Höhe) ebenso »100«. Das Gleiche wiederholen Sie mit der zweiten Uhr. Jetzt können Sie beide Uhren schön sauber mit der Maus nebeneinander positionieren (Abbildung 2.18).

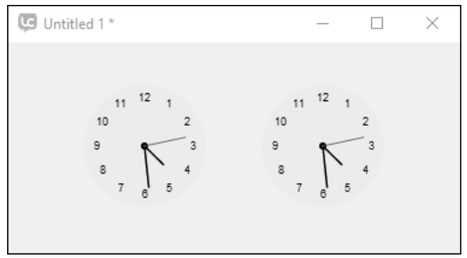


Abbildung 2.18 Beide Uhren nebeneinander in der oberen Fensterhälfte

Als Nächstes brauchen Sie zwei Beschriftungen. Ziehen Sie ein Label aus der Werkzeugpalette auf das Fenster, und platzieren Sie es unter der ersten Uhr. Gehen Sie auf die Eigenschaften des Labels, und schreiben Sie »Deutsche Zeit« in das Feld CONTENTS (der zweite Reiter im PROPERTY INSPECTOR, siehe Abbildung 2.19).

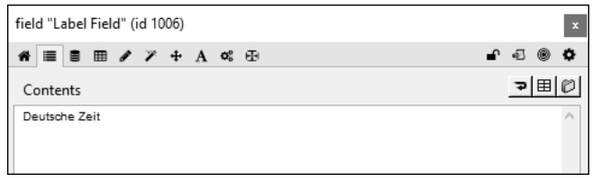


Abbildung 2.19 Der Text eines Labels wird auf der Seite »Contents« eingegeben.

Wechseln Sie nun auf die TEXT-Seite der Label-Eigenschaften, und wählen Sie dort als TEXT-ALIGN (Ausrichtung) das Symbol für »zentriert« an (Abbildung 2.20).

Abbildung 2.20 Zentrierung des Labeltextes

Nun ziehen Sie ein weiteres Label in das Programmfenster und machen mit ihm dasselbe, nur dass Sie in den CONTENT diesmal »New York Zeit« hineinschreiben. Platzieren Sie die beiden Labels jeweils unter den beiden Uhren.

Jetzt müssen noch die Uhren entsprechend gestellt werden. Dazu wählen Sie die Eigenschaften der ersten Uhr an. Wählen Sie dort auf der ersten Seite als TIMEZONE (Zeitzone) UTC+1 ATHENS BERLIN ... aus. Das sollte die aktuelle Zeit in Deutschland sein (falls nicht, wegen Sommerzeit oder so, wählen Sie eine andere Zeitzone, bis die Uhr stimmt). Die Zeitzone der zweiten Uhr stellen Sie auf die aktuelle Zeit von New York ein, normalerweise 6 Stunden vor der deutschen Zeit. Je nach Tag oder Nacht passt das Uhr-Widget sich automatisch farblich an (Abbildung 2.21).

Abbildung 2.21 Beide Zeiten nebeneinander

Wie wäre es mit einer schöneren *Hintergrundfarbe* für das Programm? Klicken Sie mit der rechten Maustaste (Ctrl + Mausklick beim Mac) in das Fenster, und wählen Sie PROPERTY INSPECTOR. Jetzt können Sie in den Eigenschaften der Seite (CARD) die Hintergrundfarbe (BACKGROUND FILL) setzen – zum Beispiel auf ein leichtes Hellblau. Die Uhren passen sich farblich entsprechend an.

Als Letztes fügen wir noch ein Extra hinzu: aktuelle Nachrichten der Tagesschau auf unserer Zeit-App. Ziehen Sie dafür das *Browser-Widget* (Erdkugelsymbol) in den unteren Teil des Programmfensters, und platzieren Sie es so, dass es die ganze Breite und Höhe, die unter den Uhren zur Verfügung steht, ausfüllt (Abbildung 2.22).

Jetzt brauchen Sie nur noch die anzuzeigende URL im Browser festzulegen. Wählen Sie dazu das Browser-Widget im Bearbeitungsmodus aus, und wechseln Sie in den Inspector (Doppelklick oder INSPECTOR-Symbol aus dem Hauptmenü).

Auf der HOME-Seite des Browsers können Sie die URL eingeben. Geben Sie hier ein: »*http://www.tagesschau.de*«

Abbildung 2.22 Zeitanzeige mit Browserfenster

Das reicht. Wenn Sie jetzt in den Laufzeitmodus wechseln, haben Sie ein fertiges, lauffähiges Projekt: eine aktuelle Weltzeituhr, die die genaue Zeit in Deutschland und New York anzeigt, darunter die Nachrichtenübersicht der Tagesschau mit den aktuellen News (Abbildung 2.23). Ein komplettes Programm – und das sogar, bevor wir mit dem eigentlichen Programmieren begonnen haben! Sie haben nur vorhandene Bestandteile von LiveCode kombiniert und mit selbst angepassten Eigenschaften versehen. Diese App könnte man bereits zu einem eigenständigen Programm für Mac oder Windows machen oder zu einer App für Android-Geräte.

Abbildung 2.23 »Hallo Welt« mal anders – eine komplette App in wenigen Minuten zusammengeklickt

Das Einzige, das vielleicht noch stört, ist der Titel oben in der Programmfensterleiste, UNTITLED 1 *. Den können Sie leicht ändern: Wechseln Sie in den Bearbeitungsmodus, klicken Sie mit der rechten Maustaste in einen leeren Bereich, wählen Sie STACK • PROPERTY INSPECTOR, und setzen Sie den TITLE auf »Weltzeituhr«. Damit haben Sie den Titel für den Hauptstack benannt – und so heißt damit die ganze App.

Am Schluss bleibt nur noch eine wichtige Sache, nämlich das Projekt zu speichern. Dazu wählen Sie im Hauptmenü FILE • SAVE AS, geben einen aussagekräftigen Namen an und speichern das Projekt an einer sinnvollen Stelle, zum Beispiel in Ihrem eigens dafür erstellten Ordner *LiveCode-Projekte*.

Obwohl LiveCode ein stabiles System ist, kann es beim Bauen von Apps schon mal dazu kommen, dass das Programm an einer Stelle festhängt oder nicht mehr reagiert. Ärgerlich, wenn man das Projekt dann nicht gespeichert hat. Daher meine Empfehlung für alle zukünftigen Projekte:

Speichern nicht vergessen!

Speichern Sie das Projekt bereits nach der Erstellung des ersten Stacks mit SAVE AS unter dem richtigen Namen an der richtigen Stelle. Während des Arbeitens sollten Sie sich angewöhnen, vor jedem Test mit Strg + S (Cmd + S beim Mac) die aktuelle Fassung zu speichern. LiveCode macht es Ihnen leicht, denn dieses Speichern lässt sich sehr gut in den Workflow integrieren, wie Sie ab Kapitel 4, »Skripte: Programmieren macht Spaß!«, sehen werden.

Kapitel 4
Skripte: Programmieren macht Spaß!

Nach dem Gestalten und Zusammenklicken von Apps soll es jetzt vor allem darum gehen, wie eine App durch Skripte mit »intelligentem Leben« erfüllt werden kann. Jetzt geht es richtig an die Grundlagen des Programmierens.

Früher bestanden Computerprogramme aus einer langen Liste von Kommandos mit allerlei komplizierten Strukturen darin. Wurde das Programm gestartet, begann es ganz am Anfang und arbeitete die Befehle dann nacheinander ab, Zeile für Zeile. Innerhalb der Liste konnte bei Bedarf hin und her gesprungen werden, und so wurden mit allerlei Aufwand Benutzereingaben verarbeitet und Ergebnisse ausgegeben. Solche Programme konnten schon mal ziemlich umfangreich werden und aus zigtausend Zeilen bestehen, je nach Verwendungszweck.

4.1 Was heißt objektorientiert?

Heutzutage wird so gut wie nur noch *objektorientiert* programmiert. Dabei hat jedes Objekt seine eigenen überschaubaren Programmskripte, die immer dann ausgeführt werden, wenn ein *bestimmtes definiertes Ereignis im Zusammenhang mit dem Objekt* eintritt. Das ist wesentlich bequemer und ermöglicht es, die Skripte viel effizienter, zielgerichteter und den heutigen Anforderungen entsprechend einzusetzen. Deshalb folgt hier eine Einführung, mit der Sie Schritt für Schritt verstehen, was es mit dem objektorientierten Programmieren auf sich hat.

Objekte und Eigenschaften

Bisher haben Sie die Eigenschaften eines Objekts immer dadurch verändert, dass Sie in den Bearbeitungsmodus gegangen sind und dann im Property Inspector die jeweiligen Werte für das Objekt eingetragen haben – und schon war das Objekt Ihren Wünschen entsprechend verändert.

Zur Erinnerung: Objekte

Objekte sind zum Beispiel alle Elemente, die wir in unserem Projekt anlegen und positionieren: Stacks, Cards, Buttons, Textfelder, Labels, Grafiken, Checkboxen, Slider usw. In jedes dieser Objekte ist ein bestimmtes Grundverhalten bereits eingebaut (Buttons können geklickt werden, Slider gezogen usw.), und die Eigenschaften der Objekte, etwa ihre Größe, Position, Farbe oder Beschriftung, können wir in LiveCode verändern.

Oder Sie haben das Objekt mit der Maus hin und her gezogen oder vergrößert und verkleinert. Das ist ja auch praktisch und sinnvoll, wenn wir die Ausgangseigenschaften für unsere Objekte setzen wollen, also die Eigenschaften, mit denen die Objekte an den Start gehen, und wie sie dann beim Aufruf des Programms erscheinen.

Wenn Sie das Programm aber später als App verbreiten, dann gibt es keinen Bearbeitungsmodus mehr – er ist dann ja auch nicht mehr sinnvoll, denn die Benutzer des Programms wollen schließlich nicht irgendwas im Property Inspector einstellen müssen. Wenn sich bei der Benutzung einer App etwas ändert (Seite, Farbe, Position, Text), soll dies natürlich an der passenden Stelle automatisch geschehen.

Das geht, weil die Eigenschaften von Objekten jederzeit durch Skripte, also durch Programme, gesetzt und verändert werden können.

Dazu können Sie jederzeit einen *Änderungsbefehl* an ein Objekt schicken, und dann wird die Eigenschaft entsprechend den Daten im Befehl verändert.

Der Befehl, um eine Eigenschaft zu ändern, lautet:

```
set (the) EIGENSCHAFT of OBJEKT to WERT
```

Wenn ich zum Beispiel die Breite von einem Button namens »meinButton« auf 200 Pixel setzen will, dann lautet der Befehl:

```
set the width of button "meinButton" to 200
```

Das ist wie simples korrektes Englisch. Ins Deutsche übersetzt also:

Setze die Breite von Button »meinButton« auf 200 (Pixel).

Der Befehl funktioniert einwandfrei, vorausgesetzt, es gibt in Ihrem Stack einen Button mit dem Namen »meinButton«.

Am besten probieren Sie das gleich mal aus. Erstellen Sie einen komplett neuen Stack Default Size, ziehen Sie einen Button in die Mitte des Programmfensters, ohne Grö-

ßen oder andere Eigenschaften zu verändern. Gehen Sie in den Property Inspector, und nennen Sie den Button »meinButton« (Abbildung 4.1).

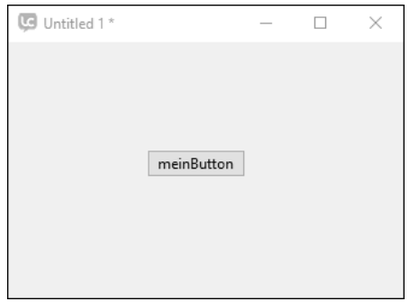


Abbildung 4.1 Ein ganz einfacher Button

Nun öffnen Sie ein Werkzeug, das Sie bisher noch nicht benutzt haben. Es dient vor allem zu Testzwecken und ist beim Entwickeln oft sehr nützlich: die Message Box. Sie öffnen sie mit Ctrl + M (Cmd + M auf dem Mac) oder durch Klick auf das dritte Symbol von links im Hauptmenü von LiveCode (Abbildung 4.2).

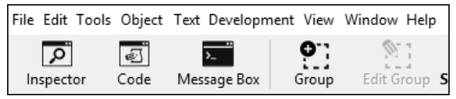


Abbildung 4.2 Das dritte Symbol öffnet die »Message Box«.

Die Message Box (zu Deutsch *Nachrichtenbox*) ist ein Fenster, das Mitteilungen an die aktuelle App senden und von ihr empfangen kann. Sie können mit ihr Befehle testen und sehen, was passiert. Sie können sich übrigens auch Ausgaben Ihrer App in ihr anzeigen lassen, was beim Test größerer Projekte noch eine Rolle spielen wird.

Schreiben Sie jetzt den vorher genannten Befehl in die obere Zeile der Message Box (Abbildung 4.3):

```
set the width of button "meinButton" to 200
```

Schließen Sie den Befehl mit der Eingabetaste (↵-Taste) ab. Und voilà! Der Button in unserem Programmfenster hat sich vergrößert. Er ist nun 200 Pixel breit.

Geben Sie jetzt folgenden Befehl in die Message Box ein (löschen Sie den vorherigen):

```
set the height of button "meinButton" to 120
```

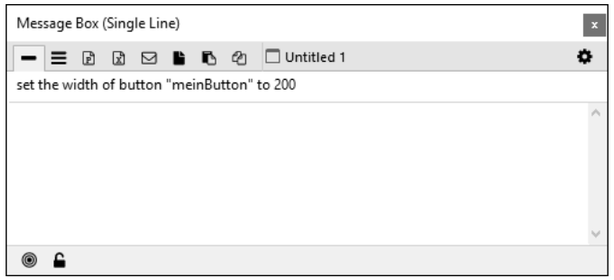


Abbildung 4.3 Die »Message Box« mit Befehl in der obersten Zeile

Nach Drücken der Eingabetaste sehen Sie, dass der Button jetzt deutlich höher geworden ist. Und wo wir schon dabei sind, probieren Sie gleich noch weitere Befehle aus:

```
set the top of button "meinButton" to 10
```

Damit stellen Sie die Position des Buttons ein, von oberen Rand aus gesehen, und so wandert der Button weiter nach oben. Auch die horizontale Position können Sie natürlich per Skriptbefehl verändern:

```
set the left of button "meinButton" to 10
```

Jetzt ist der Button näher am linken Rand. Und wie wäre es damit?

```
set the backgroundColor of button "meinButton" to "red"
```

Das verändert die Hintergrundfarbe des Buttons und macht ihn rot.

> Und wenn er dunkelrot oder hellrot oder ganz anders sein soll?

Statt der Farbnamen in Anführungsstrichen können Sie auch einen RGB-Wert einsetzen, der die Farbe mit Rot-, Grün- und Blauanteil genau definiert, zum Beispiel 123,32,55:

```
set the backgroundColor of button "meinButton" to 123,32,55
```

Die erste Zahl steht für den Rotanteil der Farbe, die zweite für den Grünanteil und die dritte für den Blauanteil. Die Werte gehen dabei jeweils von 0 bis 255. Auf diese Weise lassen sich 16 Millionen verschiedene Farbtöne definieren. Wenn Sie im Property Inspector für irgendein Objekt den Farbwähler öffnen und dort auf exakte Farbauswahl gehen (Farben definieren), dann wird Ihnen zu jeder gewählten Farbe auch ein Rot-, Grün- und Blauwert angezeigt. Experimentieren Sie gerne damit, um Ihren ganz eigenen Farbton zu finden.

Natürlich können Sie per Skriptbefehl auch die Beschriftung des Buttons verändern:

```
set the label of button "meinButton" to "Klick mich!"
```

Übrigens: Werte, die Texte sind, werden dabei so wie der Name des Objekts immer in Anführungszeichen gesetzt.

Es lassen sich alle Eigenschaften (Properties) unseres Buttons auf diese Weise per Befehl verändern, und Sie können gern noch mehr Befehle ausprobieren.

> Woher weiß ich, wie die Eigenschaften im Befehl richtig angesprochen werden?

Ganz einfach: Öffnen Sie den Property Inspector auf der Seite, wo die Eigenschaft verzeichnet ist, die Sie ändern möchten. Fahren Sie mit der Maus über die Beschriftung der Eigenschaft, und dann leuchtet die genaue Bezeichnung der Eigenschaft auf, wie Sie sie im Skript verwenden müssen (Abbildung 4.4).

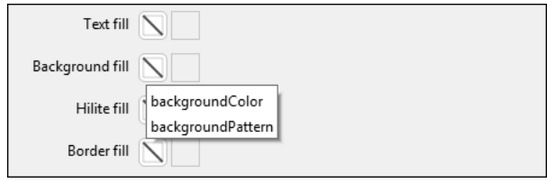


Abbildung 4.4 Die Bezeichnung von Properties, wie sie in Skriptbefehlen verwendet werden muss, erscheint beim Überfahren mit der Maus.

Manche Properties können auch nur auf *Ein* oder *Aus* gestellt werden, zum Beispiel die Sichtbarkeit. Diese wird auf `true` (wahr) oder `false` (falsch) gesetzt. So können Sie den Button unsichtbar oder sichtbar machen, der Befehl dazu ist folgender:

```
set the visible of button "meinButton" to false
```

Damit wird der Button unsichtbar (aber es gibt ihn noch!).

```
set the visible of button "meinButton" to true
```

Und damit wird er wieder sichtbar.

Selbst erstellte Eigenschaften

Und noch etwas: Sie können sogar Eigenschaften setzen, die es noch gar nicht gibt. Sie werden dann dem entsprechenden Objekt zugewiesen und können fortan jederzeit ausgelesen oder neu beschrieben werden. Diese selbst erstellten Eigenschaften heißen bei

LiveCode *Custom Properties*. Sie können sie sowohl im PROPERTY INSPECTOR anlegen als auch durch einen Skriptbefehl:

```
set the meinWert of button "meinButton" to 37
```

Damit wird die Eigenschaft `meinWert` des Buttons auf die Zahl 37 gesetzt. `meinWert` ist also eine Eigenschaft, die es im Button eigentlich nicht gibt, die aber automatisch erzeugt wird, wenn Sie etwas hineinschreiben. Danach gibt es die Eigenschaft in diesem Button, und Sie können diesen Wert (`meinWert`) auch wieder auslesen.

```
set the meinText of button "meinButton" to "Dieser Text ist jetzt im ➲
Button gespeichert"
```

Nicht nur Zahlen, sondern auch Zeichen, Wörter und Texte lassen sich den selbst erstellten Eigenschaften zuweisen.

Wozu braucht man denn selbst erstellte Eigenschaften? Immerhin bewirken sie ja nichts im Button, sie verändern keine Größe, Farbe oder Sonstiges. Sie werden diese eigenen Properties in Kapitel 8, »Multimedia, Grafik und Online-Inhalte«, sinnvoll verwenden, um Objekte zum Beispiel zu markieren oder zu kategorisieren.

Zusammenfassung: Eigenschaften setzen

Alle Properties (Eigenschaften) eines Objekts lassen sich jederzeit mit dem Skriptbefehl `set` auf beliebige Werte setzen. Die Form dazu lautet:

```
set (the) EIGENSCHAFT of OBJEKT to WERT
```

Das Setzen von Standard-Properties in einem Objekt verändert in der Regel Aussehen oder Verhalten des Objekts. Darüber hinaus kann ein Objekt auch selbst definierte Properties erhalten, in denen ein Wert oder Text gespeichert werden kann.

Was wir jetzt gemacht haben, ist aber noch nicht programmieren, sondern wir haben die Eigenschaften der Objekte mittels Befehlen verändert. Das ist vergleichbar mit »fernsteuern«. Programmieren bedeutet hingegen, dass zahlreiche Befehle miteinander in einem längeren Skript kombiniert und strukturiert werden. Erst so wird unsere App mit eigenständigem »Leben« erfüllt.

Programmieren

Die Kombination mehrerer Befehle (samt Berechnungen und Strukturen) nennt man ein *Programm*, bei LiveCode heißt das ein *Skript*. Skripte schreiben ist programmieren.

Wohin aber schreiben Sie die Skripte? Und wie, wann, warum werden sie ausgeführt? Wir müssen unserer Definition von Objekten jetzt noch etwas sehr Wichtiges hinzufügen, das wir bereits im vorigen Kapitel ausprobiert haben.

Objekte programmieren: Skripte

Objekte können neben ihren Eigenschaften auch Skripte enthalten, die bei Eintreffen bestimmter Ereignisse ausgeführt werden.

Skripte stehen also nicht irgendwo separat oder im »freien Raum«, sondern sie sind stets einem Objekt zugeordnet, oft dem Objekt, das sie auslösen soll. Wie zum Beispiel das Skript, das Sie bereits im vorigen Kapitel angelegt haben. Es war einem Button zugeordnet und begann mit der Einleitung `on mouseUp` – »falls Maustaste geklickt wurde«. Sie können Skripte aber auch der Card oder dem Stack zuordnen.

Skripte werden in LiveCode immer von einem Ereignis ausgelöst. Sie starten nicht »von selbst«, sondern es wird vorher definiert, welches innere oder äußere Ereignis dazu führt, dass das Skript gestartet wird.

Messages

Diese Ereignisse, die Skripte auslösen können, werden auch *Messages* (Nachrichten) genannt. Wenn Sie zum Beispiel im Laufzeitmodus eine Maustaste drücken, wird erst die Message `mouseDown` im System ausgelöst, sobald Sie den Button wieder loslassen, wird die Message `mouseUp` ausgelöst.

Was passiert nun mit diesen Messages? Sie schwirren nicht einfach irgendwo im System herum, sondern sie werden immer an das nächstliegende Objekt gesendet – wenn der Mauszeiger also gerade über einem Button ist, während Sie die Maustaste drücken, dann wird `mouseDown` an diesen Button gesendet. Und wenn diesem Button ein Skript zugeordnet ist, das für `mouseDown` vorgesehen ist, dann wird das Skript gestartet, und die Message wird im System wieder gelöscht. Klicken Sie hingegen auf eine Grafik, dann wird die Message an die Grafik gesendet, und wenn Sie in einen leeren Bereich Ihres Programmfensters klicken, dann geht die `mouseDown`-Message an die aktuell angezeigte Card.

Was ist, wenn es kein Skript für `mouseDown` im Objekt unter dem Mauszeiger gibt?

Dann wandert die Message weiter, eine Ebene höher, zum Beispiel zur aktuellen Card, falls es dort ein `mouseDown`-Skript gibt. Wenn nicht, dann versucht sie es noch eine Ebene

höher und prüft, ob der Stack ein Skript für `mouseDown` bereithält. Wenn auch das nicht der Fall ist, dann verhallt die Message ohne Wirkung und wird einfach wieder gelöscht (siehe dazu auch Kapitel 7).

Messages sind Ereignisse im System, die von Objekten »abgefangen« werden können. Sie können

- durch eine Aktion des Benutzers ausgelöst werden, wie den Klick mit der Maus, den Touch mit dem Finger oder das Drücken einer Taste,
- von Skripten selbst gesendet oder
- durch externe andere Vorgänge ausgelöst werden, zum Beispiel durch das Ablaufen eines Timers oder durch das Beenden eines Prozesses (Datei fertig geladen, Bild verschoben, Stack geöffnet usw.).

Aktion und Reaktion

Probieren Sie jetzt einmal die Messages `mouseDown` und `mouseUp` in der Kombination aus. Erstellen Sie ein neues leeres Projekt mit einem Stack (Größe egal). Klicken Sie jetzt rechts in das leere Programmfenster, und wählen Sie EDIT SCRIPT.

Sie sind jetzt im Skripteditor und erstellen ein Skript für die aktuelle Card. Schreiben Sie Folgendes hinein:

```
on mouseDown
```

Nach der Eingabetaste wird gleich automatisch ein `end mouseDown` hinzugefügt. Das erleichtert die Tipparbeit.

```
on mouseDown

end mouseDown
```

Zwischen diese beiden Zeilen kommt das Skript, das bei Herunterdrücken der Maustaste (über der Card) ausgeführt werden soll. Setzen Sie den Hintergrund der Karte auf Rot. Das können Sie zum Beispiel folgendermaßen machen:

```
set the backgroundColor of this card to "red"
```

> Warum `this card`?

Das `this` steht immer für die aktuelle und gerade angezeigte Karte. Sie hätten stattdessen auch den konkreten Namen der Karte einsetzen können (dazu sollten Sie ihr erst

einmal einen sinnvollen Namen geben), wie `card "meineKarte"` – Sie hätten aber statt `this card` auch `me` schreiben können. Warum? Weil das Skript ja mit der Card verknüpft ist und weil `me` ein Wort ist, das LiveCode für diesen Zweck bereithält. Sie schreiben das Skript quasi in diese Karte hinein. Dadurch »weiß« das System genau, was Sie meinen, wenn Sie `me` (»mir/mich«) schreiben.

Also sieht das gesamte Skript jetzt so aus:

```
on mouseDown
  set the backgroundColor of this card to "red"
end mouseDown
```

Probieren Sie es sofort aus: Laufzeitmodus, und ab geht's. Klicken Sie irgendwo in das Fenster. Die Fläche wird rot, sobald die Maustaste gedrückt ist. So weit, so gut. Aber sie bleibt rot. Das soll sich ändern. Fügen Sie deshalb jetzt noch folgenden Befehl im Skripteditor hinzu (hinter das erste Skript, also auch verknüpft mit derselben Card, siehe Abbildung 4.5):

```
on mouseUp
  set the backgroundColor of this card to "white"
end mouseUp
```

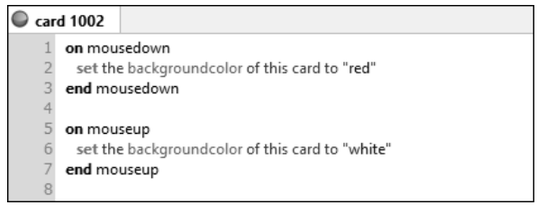


Abbildung 4.5 So sieht das gesamte Skript für die Card jetzt aus.

Wenn Sie das jetzt im Laufzeitmodus testen, passiert Folgendes: Sie drücken die Maustaste, und solange Sie sie gedrückt halten, ist die Fläche der Card rot. Lassen Sie die Maustaste wieder los, wird die Fläche weiß. Das Drücken der Maustaste sendet die Message `mouseDown` an die Card. Weil es dafür ein Skript gibt, das mit `on mouseDown` beginnt, wird es ausgeführt, und die Hintergrundfarbe der Card wird auf Rot gesetzt. Lassen Sie die Maustaste wieder los, wird eine neue Message an die Card gesendet, nämlich `mouseUp`. Auch dafür gibt es in der Card ein Skript, und das wird ebenfalls ausgeführt und setzt die Hintergrundfarbe wieder auf Weiß. Cool!

Jetzt ändern wir das Skript einmal etwas, um zu sehen, was dann passiert: Ersetzen Sie im Skript die Wörter `mouseUp` und `mouseDown` mit `keyUp` und `keyDown`, sowohl in der ersten

als auch in der Abschlusszeile der beiden Skripte. Das Skript sieht danach folgendermaßen aus:

```
on keyDown
  set the backgroundcolor of this card to "red"
end keyDown
on keyUp
   set the backgroundcolor of this card to "white"
end keyUp
```

Schalten Sie das Skript per Apply aktiv, und klicken Sie einmal mit der Maustaste in das Fenster. Nichts passiert. Klar – es gibt ja für `mouseDown` und `mouseUp` jetzt keine Skripte mehr. Drücken Sie stattdessen eine beliebige Taste auf der Tastatur: Das Fenster wird rot. Lassen Sie sie los. Es wird weiß.

Was ist passiert? Wenn Sie eine Taste auf der Tastatur drücken, wird die Message `keyDown` gesendet – und zwar an das jeweils aktive Element in LiveCode, das ist in diesem Fall die Card. Dort befindet sich jetzt auch ein passendes Skript zu `keyDown`, das also sogleich ausgeführt wird. Lassen Sie die Taste wieder los, wird die Message `keyUp` gesendet, und das dazu passende Skript in der Card setzt den Hintergrund wieder auf Weiß.

Jetzt haben Sie schon vier wichtige Messages kennengelernt, die Sie häufig nutzen werden. Es gibt natürlich noch viele mehr – manche davon sind sehr speziell und auf ganz besondere Objekte beschränkt, aber andere werden relativ oft verwendet. In Tabelle 4.1 finden Sie einige häufige Messages.

Message	Bedeutung
`mouseDown`	Maustaste gedrückt
`mouseUp`	gedrückte Maustaste losgelassen
`mouseEnter`	Mauszeiger fährt über ein Objekt
`mouseLeave`	Mauszeiger verlässt ein Objekt
`keyDown`	Taste (Zeichen auf Tastatur) gedrückt
`keyUp`	Taste (Zeichen auf Tastatur) losgelassen
`rawKeyDown`	beliebige Taste gedrückt
`rawKeyUp`	beliebige Taste losgelassen

Tabelle 4.1 Einige Messages und ihre Bedeutung

Der Unterschied zwischen `keyDown` und `rawKeyDown` ist, dass Letzteres auch dann ausgelöst wird, wenn Tasten auf der Tastatur gedrückt werden, die keinem Zeichen zugeordnet sind, wie ⇧, Ctrl, Alt, F1 usw. Möchte man auf alle diese Tasten reagieren, dann verwendet man `rawKeyDown`, wenn man nur auf konkrete Zeicheneingaben reagieren will, dann verwendet man `keyDown`.

Zum Schluss noch ein Beispiel für `mouseEnter` und `mouseLeave`. Wer bereits mit Webseiten und JavaScript gearbeitet hat, kennt vielleicht Ähnliches – wer nicht, der lernt es jetzt kennen.

Beispiel: ein Button, der Angst vor Mäusen hat

Ziehen Sie einen Button auf Ihr Fenster. Schreiben Sie folgendes Skript in den Button:

```
on mouseEnter
  set the backgroundColor of me to "red"
  set the label of me to "MAUS!"
end mouseEnter

on mouseLeave
  set the backgroundColor of me to "white"
  set the label of me to empty
end mouseLeave
```

Hier haben wir jetzt die beiden Messages `mouseEnter` und `mouseLeave` abgefangen und mit Skripten versehen. `mouseEnter` wird ausgelöst, wenn der Mauszeiger über ein Objekt fährt; `mouseLeave` wird ausgelöst, wenn der Mauszeiger ein Objekt verlässt. Diesmal sind die Skripte ein bisschen länger. Sie bestehen aus zwei Befehlen, die nacheinander stehen. Beide werden dann in dieser Reihenfolge unmittelbar nacheinander ausgeführt. (Das geht so schnell, dass es aussieht wie gleichzeitig.)

Aktivieren Sie das Skript mit Apply, und testen Sie. Fahren Sie mit der Maus über den Button, steht dort plötzlich »MAUS!«, und der Button färbt sich rot (Abbildung 4.6). Verlassen Sie den Button wieder mit dem Mauszeiger, dann wird der Button weiß, und es steht wieder nichts drin bzw. das Wort »Button«, denn so heißt der Button. Seinen Namen zeigt er dann an, wenn er keinen Wert im Label stehen hat.

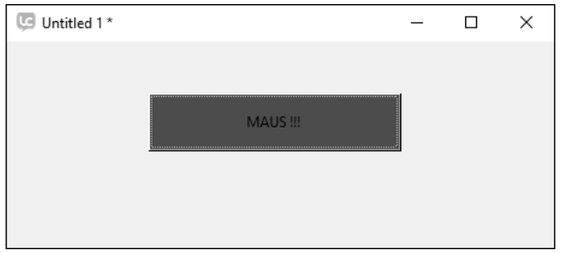


Abbildung 4.6 Der Button hat Angst vor dem Mauszeiger.

Fassen wir das Gelernte also kompakt zusammen.

Zusammenfassung: Objekte, Eigenschaften, Skripte und Messages

Ein Objekt hat nicht nur Eigenschaften, sondern es kann auch *Skripte* enthalten, die wir dem Objekt hinzufügen. Diese Skripte sind kleine Programme, die mit dem Objekt verknüpft sind. Sie werden automatisch ausgeführt, wenn das Ereignis eintritt, auf das sich das Skript bezieht.

Bestimmte eintreffende Ereignisse wie Mausklicks, das Drücken von Tasten, interne Vorgänge usw. werden vom LiveCode-System in *Messages* umgewandelt, die an das nächstliegende Objekt gesendet werden. Wenn dieses Objekt ein Skript enthält, das diese Message behandelt, wird das Skript ausgeführt.

Eine Sache noch: Viele Objekte enthalten schon ein paar eingebaute Skripte, die man *Methoden* nennt. Methoden erleichtern manche Standardvorgänge. Sie können mit einem Befehl aufgerufen werden. So kann man zum Beispiel die Methode `hide` (= verstecken) auf alle sichtbaren Objekte anwenden. Anstatt die Sichtbarkeit des Objekts auf `false` zu setzen, können Sie mit dem Befehl

```
hide button "meinButton"
```

den Button direkt unsichtbar machen.

Mit dem Befehl `show` (= zeigen) wird er dementsprechend wieder sichtbar, also:

```
show button "meinButton"
```

Schon ist er wieder da. Das nur als Beispiel. Es gibt noch zahlreiche weitere Methoden, die bestimmte eingebaute Funktionen für bestimmte Objekte ausführen.

4.2 Aufbau von Skripten

Sie haben jetzt bereits mehrere Skripte geschrieben. Sie waren alle leicht unterschiedlich, aber sie hatten auch alle etwas gemeinsam: Sie bezogen sich alle auf ein eingetretenes Ereignis (eine Message, die an das Skriptobjekt gesendet wird), und sie hatten einen oder mehrere Befehle, die in dem Fall ausgeführt werden sollen. Sie hatten auch einen Abschluss mit »end«.

Die meisten Skripte, die wir am Anfang schreiben, beziehen sich direkt auf ein Ereignis, das von außen kommt und vom System als *Message* gesendet wird – eben zum Beispiel ein Mausklick, ein Tastendruck, eine Touch-Geste usw. Schließlich ist das auch die Art und Weise, wie ein Benutzer mit einer App interagiert. Er drückt oder tippt auf eine Taste oder ein Bild und erwartet, dass die App dann etwas macht.

Später schreiben wir auch Skripte, die nicht direkt von einem Systemereignis gestartet werden, sondern die bewusst an dieser oder jener Stelle von einem anderen Skript aus gestartet werden. Aber auch diese werden durch eine Message gestartet, nur dass diese Message von einem Skript selbst gesendet wurde und nicht vom System.

Alle normalen Skripte in LiveCode sind kürzere oder längere Programme, die auf eine Message reagieren, sei es eine Message vom System oder von einem anderen Skript.

Das Gerüst eines Skripts

Grundsätzlich hat jedes Skript den folgenden Aufbau:

Element	Inhalt
on MESSAGE	Einleitung: Für welches Ereignis ist das Skript gedacht?
Anweisungen, Befehle	Hier stehen alle Befehle, die ausgeführt werden sollen, wenn das Ereignis eintritt. Das kann eine Zeile sein, es können aber auch viele Zeilen sein.
end MESSAGE	Ereignisskript endet hier.

Tabelle 4.2 Grundaufbau eines Skripts

Typische Systemereignis-Skripte haben Sie bereits kennengelernt:

```
on mouseDown
   set the textColor of the field  "text1" to "green"
end mouseDown
```

Ins Deutsche übersetzt:

Wenn die Nachricht »Maustaste gedrückt« hier ankommt,
setze die Textfarbe vom Feld »text1« auf »Grün«.
Ende des Maustaste-gedrückt-Skripts

Was machen wir aber, wenn wir möchten, dass unsere App gleich nach dem Start etwas tut – zum Beispiel einen Film starten oder auf eine Startseite wechseln oder eine Datei laden oder Werte auf Grundeinstellungen setzen oder was auch immer? Wir möchten also nicht, dass der Benutzer erst etwas machen oder klicken muss, sondern es soll sofort nach dem Start der App losgehen.

Dafür gibt es auch eine Message, die automatisch beim Öffnen und Starten einer App ausgelöst und an den Stack geschickt wird. Sie heißt `openStack` (Stack wird geöffnet). Diese Message werden Sie sehr häufig verwenden. Fast jede App muss irgendetwas machen, wenn sie gestartet wird.

Sie schreiben also das Skript zum Start des Programms in den Stack:

```
on openStack
   go to card "Startseite"
end openStack
```

Sobald die App gestartet wird, öffnet sich von selbst der Hauptstack Ihrer App (sehr viele Apps haben sowieso nur einen Stack), und das Öffnen des Stacks löst automatisch die Message `openStack` aus – diese wird vom Skript im Stack abgefangen, und das Skript wird ausgeführt. In diesem Fall wird also automatisch die Card mit dem Namen »Startseite« angezeigt, wenn die App startet.

Zusätzlich zu den vielen System-Messages, die Sie abfangen können, können Sie aber auch selbst definierte Messages verwenden, zum Beispiel das folgende Skript, das Sie einfach mal in den Stack oder auch in die Card schreiben könnten:

```
on labelLoeschen
    hide field "meinText"
end labelLoeschen
```

Dieses Skript macht Folgendes: Es versteckt das Label »meinText«. Das Skript ist kein Systemskript, sondern ein Skript, das auf eine eigene, selbst ausgedachte Message reagiert. Die Message `labelLoeschen` gibt es natürlich nicht im System.

> Wie wird dieses Skript dann gestartet?

Ganz einfach: indem ein anderes Skript, das zum Beispiel vom System gestartet wird, die Message `labelLoeschen` sendet.

Wenn Sie jetzt noch einen Button dazunehmen und in diesen folgendes Skript schreiben würden

```
on mouseUp
  labelLoeschen
end mouseUp
```

dann würde das Klicken des Buttons dazu führen, dass das Label »meinText« unsichtbar wird. Warum? Weil das Loslassen der Maustaste eine `mouseUp`-Message an den Button schickt. Das zugehörige Skript im Button wird ausgeführt und schickt eine neue Message los, nämlich `labelLoeschen`.

Für `labelLoeschen` findet das System kein Skript im Button, sucht dann eine Ebene höher in der Card und findet dort das passende Skript. Es wird ausgeführt, und das Label wird unsichtbar. Fertig.

Das ist bereits ein Mini-Beispiel für ineinander verschachtelte Skriptaufrufe. Spätere, größere Projekte werden möglicherweise aus Hunderten kleiner Skripte bestehen, die jeweils miteinander kommunizieren und sich teilweise gegenseitig aufrufen. Aber keine Sorge: Die Projekte werden stets Schritt für Schritt aufgebaut, und sie bleiben, wenn man sauber arbeitet, immer übersichtlich. Je klarer und bewusster das Projekt von Anfang an aufgebaut wird, desto einfacher wird die spätere Ausarbeitung.

Die LiveCode-Sprache: Sounds like English

LiveCode ist nicht nur der Name des Entwicklungssystems, LiveCode heißt auch die Sprache, die für die Befehle im System verwendet wird. Sie haben bereits einige Befehle kennengelernt und sie angewandt. Wenn Sie ein bisschen Englisch können, fällt Ihnen natürlich auf, dass *LiveCode* im Grunde wie simples Englisch klingt und weniger wie eine mathematische Gleichungs- und Beschreibungssprache anderer Programmiersysteme.

Das ist gewollt und ist das Prinzip von LiveCode. Befehle sollen so nah wie möglich an der natürlichen Sprache sein, damit immer intuitiv deutlich bleibt, was sie bewirken sollen. Selbstverständlich ist das für englische Muttersprachler noch einen Tick einfacher und natürlicher, aber auch mit Deutsch als Muttersprache können Sie viel einfacher in LiveCode einsteigen als in eine abstraktere Programmiersprache, selbst wenn Sie kein Englisch-As sind. Das, was Sie brauchen, ist so einfach, dass es Ihnen schnell in Fleisch und Blut übergehen wird.

Es gibt in LiveCode wie in praktisch allen anderen höheren Programmiersprachen verschiedene Sorten von Anweisungen. Es gibt Befehle, die direkt etwas bewirken: zum Beispiel der `set`-Befehl, den Sie bereits verwendet haben. Damit können Sie eine Eigenschaft eines Objekts auf einen Wert setzen. *Set* ist Englisch und bedeutet: setzen, einstellen. Und genau das macht man damit auch. Ebenfalls sehr viel verwendet wird in LiveCode der `put`-Befehl. Damit weisen Sie einer Variablen einen Wert zu oder tun sie in einen (virtuellen) Container. *Put* bedeutet: (hinein-)legen, hineintun – auch das passt. Und analog gibt es auch noch den `get`-Befehl. *Get* bedeutet: bekommen, nehmen.

Neben den Befehlen gibt es in LiveCode auch noch *Funktionen*. Damit kann man Werte berechnen, umwandeln oder bearbeiten. `round` ist zum Beispiel so eine Funktion. `round` heißt runden. `round (1.997)` ergibt 2. Die Funktion `round` rundet also eine Zahl auf die nächste ganze Zahl auf oder ab. Und so gibt es noch viel, viel mehr Funktionen, nicht nur mathematische. Sie werden viele davon in den nächsten Kapiteln bei der praktischen Arbeit kennenlernen und auch lernen, eigene Funktionen zu erstellen.

Und – für das ernsthafte Programmieren besonders wichtig – neben den Befehlen und den Funktionen gibt es in LiveCode natürlich auch noch *Strukturanweisungen*. Zum Beispiel kann man Bedingungen definieren: Bei Bedingung A wird die nächste Zeile ausgeführt, bei Bedingung B wird sie übersprungen usw. Und es gibt sogenannte Schleifen: Führe die folgenden drei Zeilen 100-mal aus oder so lange, bis alle Eingaben umgerechnet sind. Strukturen werden ausführlich in Kapitel 6, »Programmieren mit Struktur«, behandelt.

Was muss ich als Programmierer also wissen, wenn ich die Sprache LiveCode lernen will?

LiveCode

LiveCode ist eine einfache, aber leistungsfähige Programmiersprache, die im Aufbau und in der Begrifflichkeit an simples Englisch angelehnt ist. Sie besteht wie die meisten klassischen Programmiersprachen aus Befehlen, Funktionen und Strukturelementen. Mit den Befehlen wird direkt etwas bewirkt: Es werden Ein- und Ausgaben gemacht, das Programm agiert. Mit Funktionen werden Berechnungen, Umwandlungen usw. angestellt. Mit Strukturanweisungen werden Regeln für die Ausführung des Skripts gesetzt – hauptsächlich durch Bedingungen (bedingte Ausführung) und Schleifen (mehrfache Ausführung).

Füllwörter

In LiveCode klingt fast jede Anweisung wie eine Art Alltags-Englisch. Das geht so weit, dass auch bestimmte Füllwörter wie »the« und »to« erlaubt sind, die eigentlich für die Funktion nicht nötig sind. Sie haben es selbst schon kennengelernt:

```
set the color of button "mein Button" to "red"
```

Setze die Farbe des Buttons »meinButton« auf »Rot«.

Der Artikel `the` (= der, die das) ist hier eigentlich nicht unbedingt nötig. Er macht nur zusätzlich deutlich, dass es sich um eine festgelegte und bekannte Eigenschaft oder ein bekanntes Objekt handelt. Das Wörtchen `the` darf nur vor Properties oder Objekten stehen. Es würde meist aber auch ohne `the` funktionieren, solange die Anweisung eindeutig bleibt:

```
set color of button "mein Button" to "red"
```

Das geht auch und funktioniert in LiveCode – aber mit dem Wörtchen `the` fühlt sich die Sprache noch etwas natürlicher an. So würde man es eben auf Englisch sagen. Außerdem hebt der Artikel hervor, dass es sich um eine Eigenschaft des Objekts handelt.

Auch andere Befehle klingen sehr natürlich:

```
put "Hallo" into the field "Begruessung"
```

Tu »Hallo« in das Feld »Begruessung«.

Da muss man gar nicht erklären, was der Befehl bewirkt. Er erklärt sich schon recht gut selbst.

Ein anderes Beispiel, das wir schon verwendet haben, ist Folgendes:

```
hide the button "meinButton"
```

Verstecke den Button »meinButton«.

Manchmal muss man ein Objekt genauer spezifizieren, damit das System weiß, welches man meint. Zum Beispiel wenn man auf einen Button zugreifen will, der sich auf einer anderen Karte befindet als der gerade sichtbaren.

```
hide the button "mein Button" of the card "Hauptseite"
```

Verstecke den Button »meinButton« der Card »Hauptseite«.

Und man kann da beliebig verschachteln, zum Beispiel:

```
put the label of the button "meinButton" into the text of the field ➲
"meinTextfeld"
```

Ins Deutsche übersetzt:

Tu die Beschriftung des Buttons »meinButton« in den Text des Feldes »meinTextfeld«.

Ja, auch so etwas funktioniert. Mit `put` kann man Inhalte in dafür geeignete Objekte (oder in Variablen) »hineinstecken«. Das funktioniert zum Beispiel mit Text und Feldern. Man kann Text einfach per `put` in ein Textfeld »reintun«. Das ist eine Vereinfachung – man muss also nicht unbedingt die Eigenschaft `text` per `set`-Befehl setzen, sondern man steckt den Text einfach in das Feld. Damit werden wir häufig arbeiten, sodass es viele Beispiele dafür geben wird.

In der Kürze liegt die Würze

Programmierer sind faul. Oder anders gesagt: Sie lieben effiziente Ressourcennutzung. Warum also unbedingt mehr tippen als nötig? Wenn LiveCode auch die Möglichkeit bietet, in ganzen englischen Sätzen zu schreiben, so werden Sie später immer die kürzeste Form verwenden wollen. Das funktioniert auch, ist im Grunde genauso übersichtlich und erspart Ihnen einiges an Tipparbeit. Es empfiehlt sich also, den Artikel `the`, wo er nicht unbedingt zum Verständnis erforderlich ist, wegzulassen.

Auch die Namen der am häufigsten benutzen Objekte darf man abkürzen, in der Regel mit drei Buchstaben.

So kann man `btn` statt `button` schreiben, ebenso `fld` für `field`, `grc` für `graphic` oder `img` für `image`.

Zum Beispiel können Sie statt

```
hide the button "meinButton"
```

einfach schreiben:

```
hide btn "meinButton"
```

Das geht schneller, ist kürzer und dennoch sofort zu verstehen.

Oder statt

```
set the backgroundColor of the field "meinFeld" to "red"
```


könnte man auch schreiben:

```
set backgroundColor of fld "meinFeld" to "red"
```

Manche Programmierer behalten das `the` zumindest immer vor Properties (Eigenschaftsnamen) bei – was ich auch in diesem Buch so halten werde. Das hat den Vorteil, dass das System dann immer weiß, dass es sich hier um eine Eigenschaft des Objekts handelt (und es kann zur Unterscheidung wichtig werden, falls es einmal Variablen gibt, die den gleichen Namen haben). Also:

```
set the backgroundColor of fld "meinFeld" to "red"
```

Immer noch kürzer, immer noch gut lesbar. Sie werden sich sehr schnell an diese verkürzte Schreibweise gewöhnen.

Der Editor: schön sauber und farbig

Der *Editor* in LiveCode ist übrigens mehr als nur ein simples Eingabefenster. Sie können im Editor oft bereits sehen, ob Sie sich vertippt haben und wo möglicherweise Fehler sind. Dafür zeigt der Editor ein farbiges sogenanntes Syntax-Highlighting – das heißt, er markiert Strukturen, Schlüsselwörter usw. mit unterschiedlichen Farben, Schriftgraden und Einrückungen. So werden der Anfang und das Ende eine Ereignisskripts zum Beispiel immer mit den fett wiedergegebenen Ausdrücken **on** und **end** markiert. Standardkommandos wie `put`, `get`, `go`, `open`, `show`, `hide` usw. werden immer fett und olivgrün markiert. Wenn ein Kommando mal dünn und schwarz bleibt, sollte Sie das gleich misstrauisch machen, dann sollten Sie überprüfen, ob es richtig geschrieben wurde. Sonst sind die Probleme beim Ausführen des Skripts im wahrsten Sinne des Wortes »vorprogrammiert«.

Objektbezeichnungen wie `card`, `button`, `field`, `stack` usw. erscheinen in dünner Schrift mit der Farbe Lila. Wenn alles in Ordnung ist jedenfalls. Und Texte dürfen niemals leuchtend rot angezeigt werden. Ist das doch der Fall, dann zeigt uns der Editor unmissverständlich, dass da vermutlich irgendwo Anführungszeichen fehlen – meist am Ende des Textes. Mit richtigen Anführungszeichen erscheint Text blau – Objektnamen, Messages, Variablen usw. erscheinen in seriösem Schwarz und nicht fett. Wer eine Weile mit diesem Editor gearbeitet hat, gewöhnt sich irgendwann an Farbe und Formatierung, sodass es wirklich gleich intuitiv auffällt, wenn da etwas nicht stimmt.

Eindeutig fehlerhafte oder unvollständige Zeilen (Syntaxfehler) werden in LiveCode 9.5 übrigens bereits bei der Eingabe im Editor mit einem roten Icon links neben der Zeile als »Fehler« markiert – zumindest dann, wenn die Zeile klar als Fehler erkennbar ist. Das Programm lässt sich dann gar nicht erst ausführen. Aber Achtung: Nicht jeden Fehler

kann LiveCode immer erkennen und kennzeichnen – manchmal entstehen durch Fehler auch »gültige Kommandos«, die nur etwas anderes bewirken als man wollte.

Auch die Strukturen des Skripts werden durch Einrückungen im Editor gekennzeichnet. Wir haben zwar noch keine komplexeren Strukturen behandelt, aber zumindest haben wir eine Einleitungszeile, das eingerückte eigentliche Aktionsskript und dann wieder die ausgerückte Endzeile.

»Moment mal«, werden Sie vielleicht erwidern, »bei mir sieht das aber gar nicht so schön sauber strukturiert aus, nur wenn ich es selbst mache, wird es eingerückt.« Das stimmt – es gibt einen Trick, den man kennen muss.

Tipp: Skripte formatieren auf Knopfdruck

Klicken Sie mit der Maus in irgendein Skript, und drücken Sie dann auf der Tastatur einmal die Tabulatortaste (⇥). Und schwupp! Schon ist das gesamte Skript sauber formatiert.

Diesen regelmäßigen Klick auf die Tabulatortaste werden Sie sich fest angewöhnen. Allein schon weil es immer so schön aussieht. Einmal ⇥ drücken – und alles ist schön und übersichtlich (Abbildung 4.7).

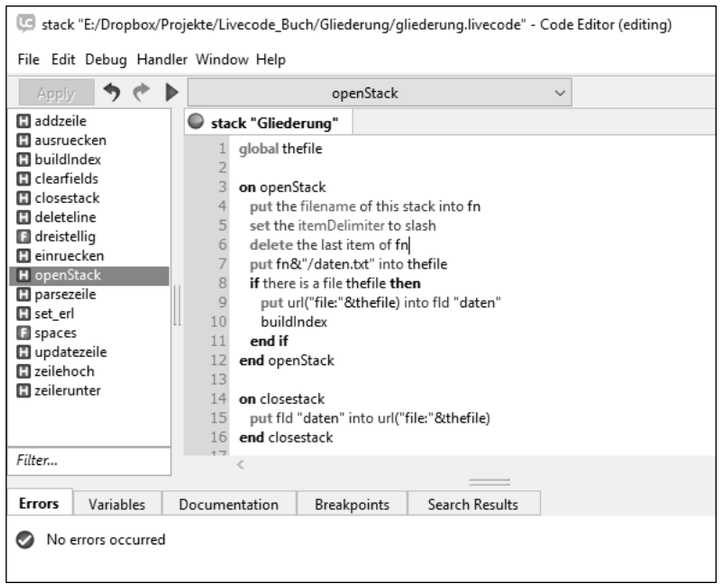


Abbildung 4.7 Der Editor in Aktion mit schönen Strukturierungen

Kommentare

Sie können im Programmcode auch jederzeit eigene Kommentare unterbringen. Die werden vom Editor dann schön sauber in Grün markiert. Einzeilige Kommentare kennzeichnet man, indem man entweder zwei Minuszeichen voranstellt oder zwei Schrägstriche:

```
-- Das ist ein Kommentar
// Das ist auch ein Kommentar
```

Und wenn ein Kommentar mehrere Zeilen umfasst, dann wird er mit /* eingeleitet und mit */ wieder beendet:

```
/* Hier beginnt ein Kommentarbereich
Immer noch Kommentar.
Hier endet er. */
```

Damit lassen sich beim Testen auch Skriptteile »auskommentieren«, wenn man sie vorübergehend weglassen möchte. Aber der Editor kann noch mehr als nur Skripttext markieren und farbig hervorheben. Er verschafft uns auch einen Überblick über die verschiedenen Ereignisskripte, die es zu einem Objekt gibt. Durch Anwahl des entsprechenden Ereignisses in der linken Spalte springen wir direkt zum passenden Skript.

Links oben hat der Editor die Apply-Taste, mit der Sie Ihre Skripte »scharf schalten«. Und wenn da etwas schon von der Syntax her nicht stimmt, also formal falsch ist oder falsch geschrieben wurde, »meckert« der Editor sofort – mit Recht. Denn dann würde es schlicht und einfach nicht funktionieren. Der Editor gibt mit den roten Icons sowie in seiner unteren Zeile unter dem Reiter Errors auch Hinweise, wo der Fehler zu suchen ist (Abbildung 4.8).

Wenn das Skript wegen eines Fehlers stoppt, können Sie es mit der grünen Play-Taste oben im Editorfenster übrigens wieder weiterlaufen lassen – oder mit dem Stop-Symbol ganz beenden.

Neben den Errors gibt es auch den sehr praktischen Reiter Documentation unter dem Skriptfenster. Dieser zeigt Ihnen immer die Dokumentation, also die Definition und Anleitung zu dem Schlüsselwort, auf dem sich der Cursor im Editorfenster gerade befindet. Wenn Sie noch mal wissen wollen, wie man einen Befehl verwendet oder wie man ein Objekt anspricht, dann klicken Sie mit der Maus auf das entsprechende Wort, und wählen Sie Documentation darunter aus (Abbildung 4.9).

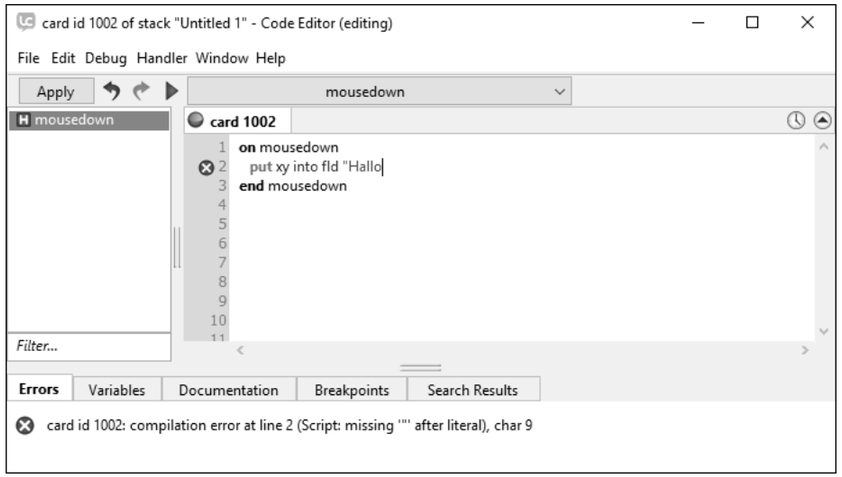


Abbildung 4.8 Der Editor »meckert« und wird rot, wenn Sie ein fehlerhaftes Skript aktivieren möchten. Hier fehlen Anführungszeichen.

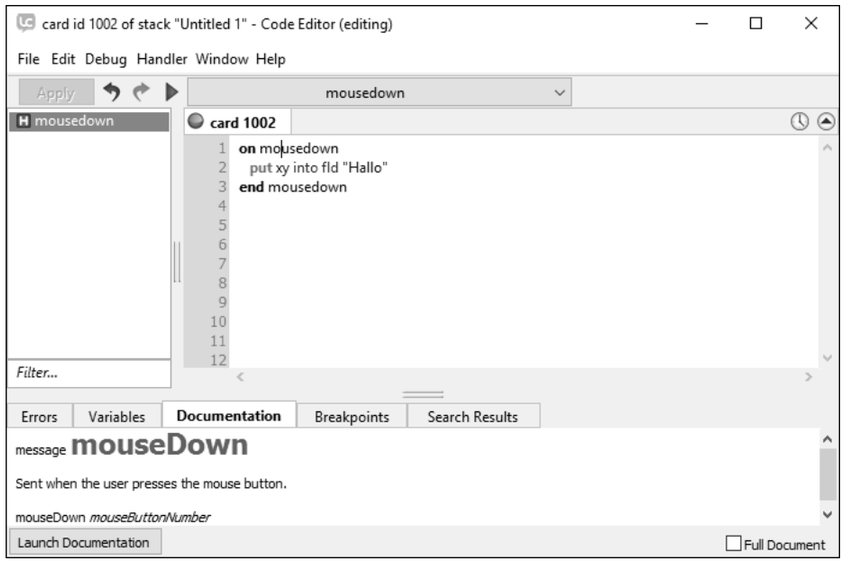


Abbildung 4.9 Dokumentation zu »mousedown«

Auf einen Blick

Inhalt