中学英語くらいの読んでわかる
プログラミング言語&統合開発環境

LiveCode

プログラミング初心者開発入門

オープンソース無料版でソフト開発
LiveCodeを開発してきたRunRev社のある
スコットランドの25%以上の高校が
コンピュータ・サイエンスの授業に採用しています。

Mac OS & Windows
デスクトップアプリを同時に作る
開発入門から中級編

第１部: ほとんど英語テキストでの開発
第２部:日本語テキストを使う開発

メディア・アーチスト
小島健治 著

詳しい目次

第一部： ほとんど英語テキストでの開発

第二部：日本語テキストを使う開発

第一部をほぼ理解したとして書いています。「日本語テキストを使う開発」を読み始める前に、特に10章をもう一度読んでください。

20：日本語（ユニコード）の基礎　..... 284

- フィールドからフィールドに文字列を移動
- 「put」はフィールド、「set」はそれ以外
- ボタンのレイベルをスクリプトで日本語に
- ランゲージからフォントを分類
- スタックに日本語タイトル。 日本語フォントでテキストをセットする
- 日本語（ユニコード）の基礎　まとめ

21：フィールドにある日本語の扱い　..... 295

- 日本語のラインやキャラクターの移動
- 日本語をクロスプラットフォームで保存する 始めはUFT8です
- 日本語をSJISで保存する
- 日本語のUTF8とSJISをインポートする
- BOMを付けたユニコードUTF16の保存

22: ボタン、メニューの日本語　..... 302

- 日本語のラジオ・ボタン
- 日本語でアラート・ウインドウを出す
- 日本語のタブ・メニュー
- 日本語のメニュー

23: 日本語（ユニコード）の実践、その他　..... 316

- 日本語のテーブル・セルを差し替え
- 日本語のメールを作る
- 日本語配列のキーボード
- 日本語のツール・チップ
- 日本語ファイル名の書類を保存
- 日本語ファイル名の書類をインポート

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

書き始めるにあたって

「LiveCode」はプログラミングの特殊な専門知識がなくても、読んで日常英語のようにわかりやすい言葉でプログラムが書ける、クロスプラットフォームの統合開発環境です。LiveCodeは初期のMacOSに付いて来たHyperCard（1987〜98 使われた言語名はHyperTalk）と言うアプリケーションから派生したUnix版の「MetaCard」を（後にWindows、MacOS用も加えられる）2003年スコットランドのRunRev社が権利を買い取り、開発ツール等に大幅な改良を加えて、始めは「RuntimeRevolution（RunRev）」と言う名称で売り出されました。RunRevはその後iOS、Android、サーバーの開発環境が加えられ、2010年に名称を「LiveCode」と変更しています。LiveCodeは、開発するアプリケーションのインターフェイスを作るツールや、編集機能を含めた開発環境であり、そのインターフェイスを機能させるプログラミング言語の名前でもあります。英語での分かりやすさから、LiveCodeを開発しているRunRev社のあるスコットランドでは、25%以上の高校がコンピュータ・サイエンスの授業にLiveCodeを採用しています。

オープンソース版の「LiveCode Community Edition」は、2013年にクラウドファンディングのKickstarterで資金調達が目標に達して生まれることになりました。オープンソース版の「LiveCode」は2013年4月10日から配布されています。LiveCodeオープンソース版は、GNU_GPLv3ライセンスの下にリリースされますから、コマーシャル、シェアウエア、フリーウエアに関わらず、LiveCodeオープンソース版で作ったソフトウエアは、同じくGNU_GPLv3ライセンスの下にオープンソースで（書かれているソースコードを公開して）配布しなくてはいけません。個人やインハウスが使用目的で開発する場合は、ソースを公表する必要はありません。詳しくはGNU Operating Systemを読んでください。日本語翻訳ページ
http://www.gnu.org/licenses/gpl-faq.ja.html

すべてのMacに無料で付いて来たHyperCardは、カードメタファと言われるコンピュータ以前の図書館の検索カードのような、整理法をベースとしたユーザーインターフェイスを採用して、暗号のようなコンピュータ言語ではない、人間の言葉（英語）に近いスクリプト言語で、プログラミングの専門家でない多くの人々にも親しみを持って受け入れられ、沢山のアプリケーションが生み出されました。コンピュータのカラー化が進み、インターネットも普及し始めた頃、アップル社は時代にあった改良をしないままHyperCardの配布も打ち切って、HyperCardの時代は終了しましたが、LiveCodeによってその思想は受け継がれ、さらに改良されて現在に至っています。ある時期には、HyperCardで使われたHyperTalkに類似する言語（ダイアレクト）を、総称してXTalkと呼んでいましたが、現在はあまりそう呼ばれる事もありません。（LiveCodeで使う言語も、MataCardの時代にはMetaTalk、RunRevの時代にはRevTalkと言う名前でした。その他のXTalkダイアレクトにSuperTalk、SenseTalk等があります）

LiveCodeで書いたプログラムは、Mac OS、Windows、Linux、iOS、Androidと、サーバー上で動かすことができます。また上記のHyperCard（HyperTalk）で使われていた言葉は、ほとんどLiveCodeでも使う事ができます。さらにHyperTalk時代にはなかった様々な時代にあった環境の

About 小島健治

コンピュータを使用した知覚認識の関連を探求するアーチスト。ニューヨーク市在住。1980年よりニューヨーク市でペインティングを制作。中世の技法・材料を現代に持ち込んだ作品が、シティバンク、ヘス石油等のコレクションに加えられる。80年代に急速に発達したパーソナル・コンピュータに興味を持ち、絵画の下図作成に使用する。マッキントッシュに付いて来たHyperCardのインターラクティブにひどく感銘。カラー化されたコンピュータ時代に、白黒しか扱えなかったHyperCard（言語名：HyperTalk）から、カラーやメディアの扱えるSuperCard（言語名：SuperTalk）にスイッチ。アート・ワークとしてのコンピュータ・ソフトを本格的に考え始める。
その頃からインターネットの時代が始まり、SuperCardのプラグインを使用したWebアートを開始。マッキントッシュ上だけのSuperCardに限界を感じ、Java Applet、Flashを使い幾つものWebアートを制作。2001年までの主なWebアートが、ニューヨーク市ニューミュジアムのアーカイブとなる。その頃試していたMetaCard（クロスプラットフォームで動くHyperCardの類似プログラム）がRuntime Revolution社から販売されることとなって、本格的にRuntime Revolution（RunRev、LiveCodeの以前の名称）を使用開始。視覚・聴覚・時間の関連を追及したソフトウエアアート「RGBミュージック」をニューヨーク市で発表。RGBミュージックは、RE-NEW、Pixelstorm、BOURGES、PROCESS、FILE PRIX、FAD、Live Herring、Istanbul Contemporary Art Museum等、ヨーロッパ、ブラジル各地で開催されたメディアアート・フェスティバルに多数選出、受賞。 kenjikojima.com
小島健治 デジタル・アート ワークス：概略

言語も加えられ、現代のソフトウエアが開発できるように改良されています。LiveCodeで書くデスクトップアプリは、基本的には一つの書類で書き進めて、配布できるスタンドアロンのアプリに仕上げる（ビルドする）時に、それぞれのプラットフォーム用に自動的に分けられて完成されます。もちろんプラットフォームによって独自の特徴がありますから、上級者になってプラットフォームのディテールに入って行くと、それぞれの違いに対処する必要が出て来て、一つの書類の中にこのプラットフォームならこうと言う、振り分けをして書かなくてはいけない場合もしばしばあります。ここではMac OSとWindows共通の「LiveCode Community Edition」でできる、デスクトップ・アプリについて書いています。スマートフォン・アプリは、プラグインで開発環境を変えたデスクトップのバリエーションですから、まずデスクトップでプログラミングできる事が基本となります。ただしオープンソース無料版のLiveCodeには、スマートフォン開発用のプラグインは含まれていませんから、スマートフォン・アプリ開発には有料版のLiveCodeを購入する必要があります。サーバーでLiveCodeを働かせるには、LiveCodeのサーバーエンジン（Server Deployment）がインストールされたサーバー、またはRunRev社が運営するOn-Revのようなプロバイダーのサーバーが必要です。

コンピュータ・プログラミングと言うと、普通理数系の強い人達が書くと思われがちですが、私が関わって来たLiveCodeの経験から言うと、決してそんな事はありません。しかし最低限、足し算と引き算、掛け算と割り算は理解していないと困ります。英語はできるに越した事はありませんが、上にも書いたように中学程度の理解力があれば、プログラムは書き進めます。上級になって特殊な調べものが必要になったりした場合、日本語のコミュニティが確立されていない現在は、ある程度の英語力があった方が良いでしょう。しかしそれは、どのコンピュータ言語でも同じかもしれません。LiveCodeのちょっとした機能の小さなプログラムでしたら、随筆とか詩歌やドローイングのような感覚でしょうか。また事柄・機能が幾つもある大きなプログラムでしたら、前章からの流れを取り込んで展開する小説を書いて行くような感覚や、土台から形作り、細部の描写をしながら、構図や色彩バランスを考えて仕上げて行く、ペインティングに近いと言えるでしょう（実際には私は小説を書いたことはないのですが）。言える事は機能が多い少ないに関わらず、全体の骨格を見通せる少し覚めて離れた視点と、細かなディテールの正確な描写が必要です。しかし学習の始めから全体を見通しながら、細かな部分を描いて行くのは不可能ですから、まず部品としてのインターフェイスの扱い、各部品（オブジェクト）の持っている特性（プロパティ）や、その操作法の幾つかの慣用的な言い回し（プログラミング）に慣れて行くのが大事かもしれません。言葉自体は日本人でも意味の分かる簡単な英単語の集積ですから、いくつか文章を書き進むにしたがって、プログラミングの構図も徐々に見えて来るでしょう。

この文を書き始めて、15世紀始めにイタリアの画家チンニーノ・チンニーニの書いた絵画技法書「Il libro dell'arte 芸術の書（英訳：The Craftsman's Handbook）」を少し思い出しています。「芸術の書」は、中世から初期ルネサンスの絵画材料と技法を知る上で重要な本です。チンニーニは技法以外に、絵画を描く上での生活の態度や心構えについてのアドバイス、神学や哲学などを学ぶ事の大切さ等にしばしば触れています。勿論その当時絵画を描く事は、神の姿を具現化することで、彼は本の始めに、神、聖母、セイント達に、畏敬の念を表す事から書いています。21世紀に住む私たちは、その頃とはまったく違う価値観のもとに、プログラミング技法書を読む訳ですが、人間の言葉により近いコンピュータ言語の背後にある思想がいったい何なのか、単純に現在を生き抜く実用書なのか、それとも大きな歴史の方向を示そうとするモノなのか、書き進むにしたがって、私自身の考えも整理されればと思っています。

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

こんな人にLiveCodeをお勧めします

- 「暗号のようなプログラミングを覚えるのはぜんぜん興味ないけれど、プログラミングと言うのがどういうものなのか、自分でコードを書いていろいろ遊んでみたい。」
- 「初めてプログラミングを学ぶのだけれど、言葉の種類があり過ぎて何を選んだら良いかわからない。」日常的な簡単な英語程度の読んで理解できるLiveCodeは、日本の中高生でもプログラミングの考え方や、ソフトウエアの作り方を学ぶには最適です。
- 「中高生向きのプログラミング学習教材を探していた。」LiveCodeのコミュニティー・エディションは無料で、しかも言語は簡単な日常英語をもとに作られています。LiveCode開発元のRunRev社のあるスコットランドでは、25パーセント以上の高校のコンピュータ・サイエンスの授業に採用されています。
- 「アートワークをソフトウエアで構築して、そこから生み出す独自の作品を作りたい。」著者小島健治(アーチスト)のソフトを紹介します。「RGB MusicLab」「時間系不定時報」
- 「定年退職したので、今までできなかったおもしろそうな新しい事をやってみたい。」
- 「ゲームが好きで遊んでいるのだけれど、自分で考えたゲームを作ってみたい。」
- 「プログラミングに少し興味はあったので、有名言語を幾つかトライしてみたけれど、どうもとっつきにくく諦めてしまった。」プログラミングと言うと理数系の強い人というイメージがありますが、LiveCodeは文科系、アート系の人にも勧めます。
- 「個人・学校・社内だけの、目的がシンプルなアプリケーションが欲しいのだけれど、適当なものを売っていない。」LiveCodeの統合開発環境を使えば、ソフトの基本構造を短時間で仕上げる事ができます。さらにLiveCodeの言語ではカバーできない特殊な機能は、学内、社内にC++の協力者がいれば、エクスターナルとして取り込むことができます。エクスターナルの開発キットが用意されています。
- 「HyperCardや類似のアプリケーションを使った事がある。」HyperCardの言語HyperTalkのほとんどはLiveCodeでも使う事ができますから、すぐにアプリケーションが作れます。さらにLiveCodeには、現代のアプリケーションを開発する為の言葉が追加されています。
- 「売っているようなアプリを、クロス・プラットフォームで作って配布したい。」LiveCodeは簡単な言語と言っても、市販のアプリと比べて遜色のない製品を作る事ができます。フリーウエア、シェアウエアだけでなく、コマーシャル版として作ったアプリを売り出す事も可能です。オープンソース版はGNU GPLv3ライセンスの下にリリースされますから、配布のアプリはソースコードを公開しなくてはいけません。
- 「デスクトップ・アプリだけでなく将来的にスマートフォンやサバー・アプリも作りたい。」無料版オープン・ソース「LiveCode Community Edition」には、スマートフォン開発のプラグインが含まれていません。スマートフォン開発には、LiveCodeのコマーシャル・ライセンスが必要です。

日本語の本格的コマーシャルアプリを作りたい人にはお勧めしません。
他の言語でプログラムを書いてください。

- LiveCodeで作るプログラムは多言語対応（ユニコード）で、日本語も自由に扱えます。しかし日本語OS上では、以下の問題があります。１）ファイル名を日本語にしたり、作ったアプリケーションを日本語名フォルダーに入れると起動できません。別な言葉で言うと、ファイルパスに英数字以外は使えません。2）開発時のプログラム中に日本語のコメントが書けません。コメントは英語かローマ字で書きます。またコマーシャルアプリを作るのでしたら、オープンソース版でなくコードを公開しないコマーシャル版のLiveCodeの方が適切かと思います。

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

01: ダウンロード、インストール

この章の概略

- *「無料オープンソース版 LiveCodeコミュニティ・エディション」の、ダウンロードからインストールまでです。表示が英語と言うだけで、通常のアプリケーションのインストールと変わりありません。MacOSとWindowsもほぼ同じです。*
- *無料オープンソース版とコマーシャル版の違い。*

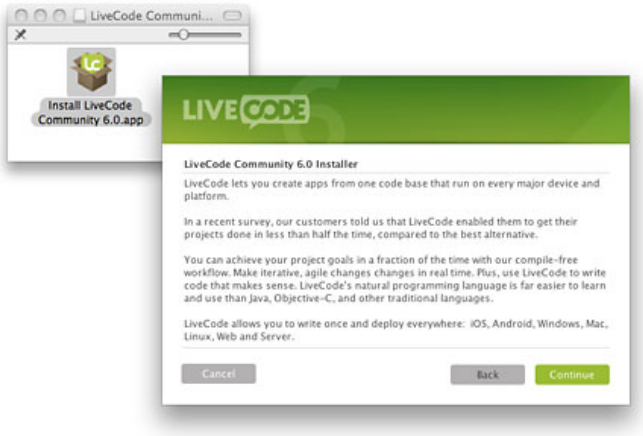
LiveCodeのサイトに行ってインストーラーをダウンロードします。

http://livecode.com/download/

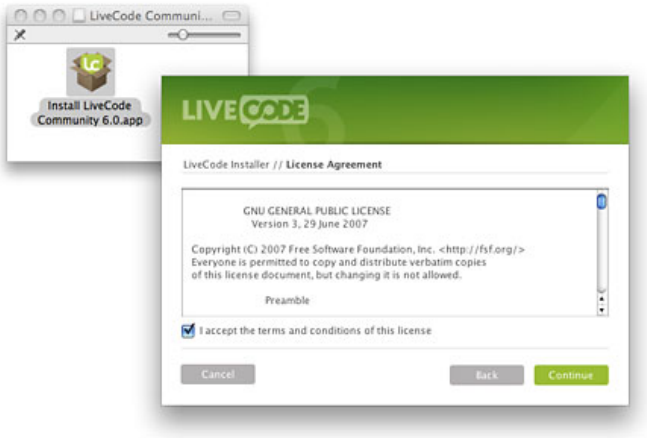
MacOS版インストーラ：Install LiveCode Community 6.1.app

Windows版インストーラ：LiveCodeCommunityInstaller-6_1_0-Windows.exe

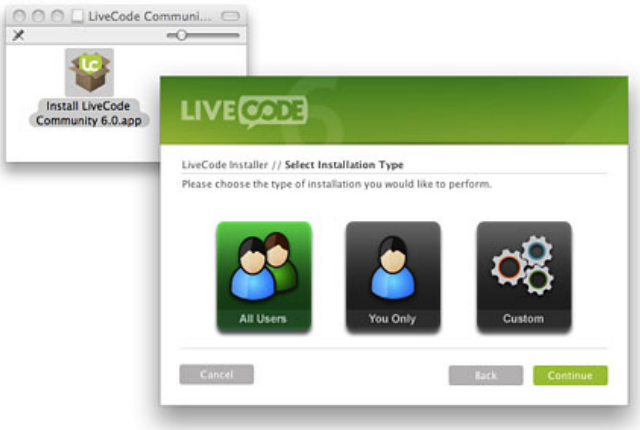
以下の図は LiveCode Community Edition 6.0 MacOSバージョンです。バージョンが上がった段階で、多少の違いがあるかもしれません。インストーラのダブルクリックで



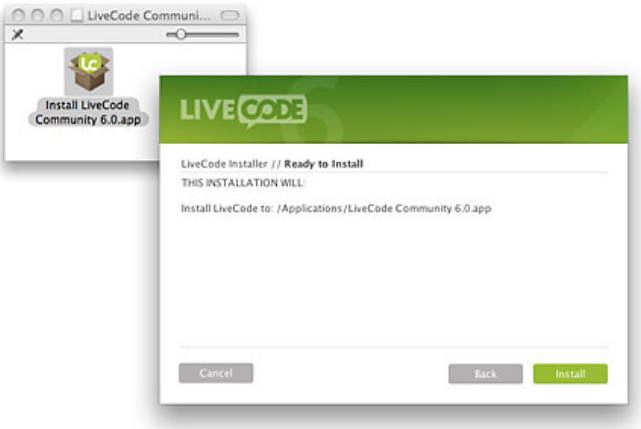
LiveCodeコミュニティ・エディションの短い説明が出ます。右下「Continue」をクリックすると、次のページ「License Agreement」が出ます。



同意できる場合は、左下「I accept terms and conditions of this license」にチェックマークを付けて、右下「Continue」をクリックします。

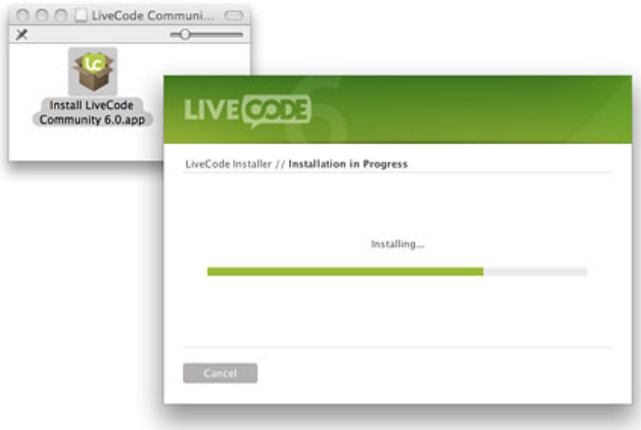


どこにLiveCodeをインストールするか、インストールのタイプを選びます。ほとんどの場合はデフォルト設定の「All Users」で良いでしょう。右下「Continue」で続けます。

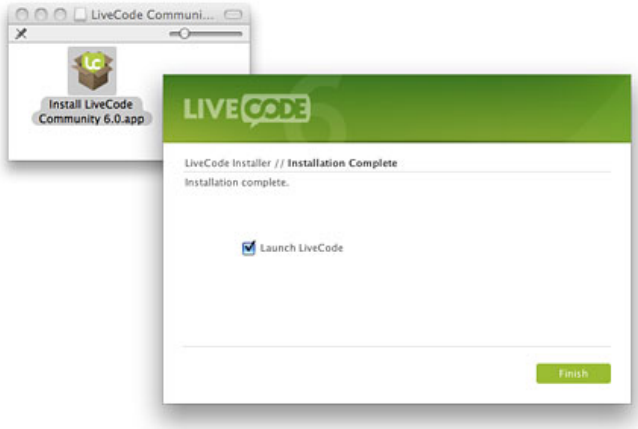


インストールの準備ができました。どこにインストールするかインストール先が表示されます。インストール先を変更したい場合は「Back」で前のページに戻ってインストールのタイプを変更します。「アプリケーション」以外の日本語名のフォルダーの中にLiveCodeをインストールすると、LiveCdeは起動できません。インストール先を確認して右下「Install」をクリッ

クします。



しばらく待っていると



画面が変わってインストールが終わります。すぐにLiveCodeを開きたいときは「Launch LiveCode」にチェックを入れたまま「Finish」をクリックします。LiveCodeアプリが起動されて

始めのスプラッシュ・ウインドウがしばらく開かれて、メニュー等が表示されたあと

LiveCodeを起動させるための、ダウンロードの時に登録したメールアドレスとパスワードを入れ「Activate」をクリックして、LiveCodeアプリが使用できるようになります。

無料コミュニティ版と、有料コマーシャル版の違い

LiveCodeはもともとRunRev社が開発して売り出している、アプリケーションの統合開発環境です。RunRev社がLiveCodeを、コミュニティ版と言う名前でオープンソース無料版にした後も、コマーシャル版はRunRev社から売り出されています。

- コミュニティ版は、開発したプログラムのコードを開示しなくてはいけません。コマーシャル版はコードを見せない設定にできます。デスクトップ・アプリの開発機能上はこの違いだけです。スマートフォンの開発プラグインは、コミュニティ版に含まれていません。
- コミュニティ版は無料です。コマーシャル版は有料（年間すべてのアップデート330ポンド。2013年）です。

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

02: 創造するコマンド「Create」

この章の概略

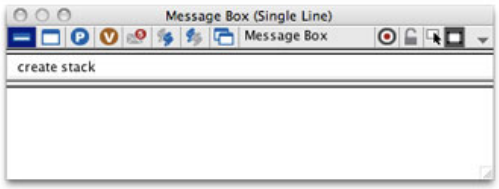
メッセージ・ボックスからプログラミング言語LiveCodeeの命令文をコンピュータに送る。
ここではインターフェイスの開発ツールを用いません。

- *アプリケーション開発の土台となる「スタック」その他のオブジェクトを作成。*
- *メッセージ・ボックスからスタック内のカードを移動させる。*

始めにスタックありき

私たちはこれからLiveCodeと言うコンピュータと対話できる言葉を使って、少しづつ目に見えるオブジェクトや、その働きを作って行きます。始めにLiveCodeの言葉を使ってコンピュータにメッセージを送る、メッセージ・ボックスで交信してみましょう。LiveCodeのアプリケーション・アイコンをダブルクリックでオープンすると、デスクトップの上方横に並んだメニュー・バー、左に上下に並んだツール・パレットと、何もないデスクトップの空間だけが見えます。あるいは「Project Browser」と「Application Browser」のウインドウが見えているかもしれません。今はこのふたつのウインドウは閉じてください。

メニューバーには、トップに文字だけの「File」「Edit」など通常のアプリ用のメニューの下に、アイコンで「Inspector」「Code」「Message Box」などが並んでいます。「Message Box」をクリックします（上図赤丸）。「Message Box (Single Line) メッセージ・ボックス」が現れます。「Message Box (Single Line)」がトップにあるタイトル・バーに表示されていない時は、メッセージ・ボックスの上方に並んだアイコンの左端をクリックして選びます。それではまだ何もないこのデスクトップの空間に、新しくプログラムの世界を築き上げる大地を創ります。図のように、メッセージ・ボックス上段のテキスト・スペースに「create stack」とタイプして、リターンキーを叩きます。




```
create stack
```

スタックを創りなさい。

デスクトップ上のどこかに小さな四角形がひとつ現れます。コンピュータはプラスとマイナス

この章で使われる英単語

- create [動詞：創造する, 生み出す, 作り出す, 作る]
- stack [名詞：束になって一つにまとまった物, スタック]
- set [動詞：セットする, （ある状態に）整える]
- string [名詞：連続した文字の並び, 紐]
- card [名詞：カード]
- button [名詞：ボタン]
- go [動詞：行く, 進む]
- next [形容詞：（順序が）次の]
- previous [形容詞：（順序が）前の]
- quit [動詞：やめる, 終了する]

の電気信号でないと、理解して働いてくれませんから、人間との間にいくつものステップを経て、お互いに理解できる言葉や事柄に変換して対話を進めて行きます。私たちが理解できる英語の言葉「create stack（スタックを創りなさい）」は、少し機械に近い言葉に翻訳されて、さらにもう少し機械に近い言葉に翻訳されて、と言うステップを幾つか踏んで、電気信号になってコンピュータに伝えられ処理されて、結果はそこからまた同じ道程を逆にたどって、最後に具体的な目に見える、四角形のオブジェクトになって現れます。

今コンピュータとの対話で「create stack」と言う言葉を使いました。「create」は日本語でも「クリエイト」で分かるでしょう。もう一つの単語「stack（スタック）」は英語で、一つにまとまった束と言う意味です。何枚ものハガキやカードをきちんと揃えて、まとめた束のような物を想像すれば良いです。LiveCodeではこの四角形のオブジェクトを「スタック」と呼んで、プログラム開発の目に見える他のオブジェクト、イメージやテキスト・フィールド等の土台となるものです。これは言葉の世界で「スタック」ですが、具体的な四角形になると人間の思考で何通りもの違うモノに想像できます。私は何もない空間にスタックの「大地」を創ると言いました。見方を変えれば宇宙に漂う「惑星」とも、大劇場の「舞台」とも、映画館の「スクリーン」とも、昔ながらの図書館の「検索カード」とも、一組の「トランプ」とも、スタックの中に様々なデータを組み込んで行く事で、用途にしたがって何とでも考えられる事が可能です。さらに今はデスクトップに浮かぶ一つのスタックだけの小さな世界ですが、ここからインターネットで別の宇宙に繋がって行く事もできます。

デスクトップ上にあるスタックは、一般的なコンピュータの用途ではウインドウ（window）とも言えます。実際に

```
create window
```
ウインドウを創りなさい。

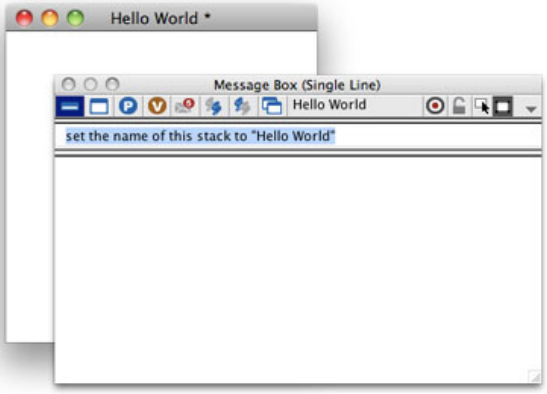
とメッセージ・ボックスから送っても同じ結果を得る事ができますが、LiveCodeの柔軟性を示すだけにして、これからこの四角形のオブジェクトは「stack」と統一して、書き進めてゆくことにします。プログラミング言語LiveCodeでは「stack スタック」は、データ（カードやバックグラウンド）を束ねるオブジェクトの構造を言います。詳しくは後の章で。

値を設定する「set」

この四角形のオブジェクトに名前を付けます。どこかデスクトップ上の隅の方にあったら、適当な処に移動させてください。今は四角形の上部には「Stack 1365539028*」のように、数字の仮の名前が入っています。伝統的なコンピュータ言語学習の作法にしたがって、「Hello World」と名付ける事にしましょう。メッセージ・ボックスに下のスクリプト

```
set the name of this stack to "Hello World"
```
このスタックの名前を "Hello World" と設定しなさい。

をタイプまたはコピペして、スペルの間違いがないかを確認したら、リターンキーを叩くと、数字に変わって「Hello World*」が四角の上部にあるバーに現れます。スペルの間違いがあるとメッセージ・ボックス下段のスペースに「Script compile error: エラー」の表示が出ますから、もう一度確認してください。



ダブルクオート「"」で括ったスタックに付けた名前 "Hello World" は、LiveCodeでは「string（ストリング）」と言っている文字列で、アルファベットか数字の普通の英語の文または単語が使えます。普通の英語の文と言ったのは、LiveCodeのプログラミングの言葉としてリザーブ（予約）されているいないに関わらず、ダブルクオート内なら何でも自由に英語の言葉・数字が使えると言う意味です。また逆に言えば、英語の表現に近いLiveCodeと言っても、ダブルクオートの外は厳密に決められた言葉と文法に従わないといけません。もしダブルクオートの外でスペルを間違ったり、対象物（この場合は「this stack」）が見当たらなかったりしたら、LiveCodeはコンパイル・エラーになってストップしてしまいます。LiveCodeでは他の機械レベルに近いコンピュータ言語と違って、数字をストリングにした場合、文章としての数字とも、値としての数字としても扱うことができます。後章で具体例が出た時に詳しく。

チュートリアルの後半まで、日本語での名前の付け方等は忘れてください。
プログラミング自体、もともと英語をベースに作られているので、
英語である程度表現ができるようになった後、可能な日本語表現を加えて行きます。

ボタンとカードを創る

スタック「Hello World」の上にボタンを一つ作ります。メッセージ・ボックスから

```
create button
```

ボタンを作りなさい。

を送りると、スタック中央に四角い横長のボタンが現れます。上のスタックの説明に「何枚ものハガキやカードをきちんと揃えて、まとめた束のような物」と書きました。今ここにあるオ

ブジェクトは、スタックがひとつ、スタックの中にカードがひとつ（１枚）、カードの上にボタンがひとつあります。このカードの上にもう１枚新しいカードを作ります。メッセージ・ボックスから

```
create card
```
カードを作りなさい。

を送ると、ボタンは見えなくなって新しいカードが現れます。スタックの上部のバーは「Hello World (2)*」となります。これはスタックのカード・ナンバーが「2」を表しています。もう一度メッセージ・ボックスのスクリプトのリターンキーを叩くと、スタックの上部のバーは「Hello World (3)*」となります。

カードの移動

スタック「Hello World」にはカードが３枚あります。現在はスタックの上部のバーは「Hello World (3)*」となっているので、次々にカードを移動してそれぞれを開いてみましょう。

```
go to the next card
```
次のカードに行きなさい

と、メッセージ・ボックスにタイプして、リターンキーを叩き続けると、次々にスタックの上部のバーのナンバーが変わって行きます。「Hello World (1)*」が開けられた時には、先ほど作ったボタンが現れ、それ以外は白地のカードが見えます。「go to the next card」は他の言い回しもできて

```
go to next card
go next card
go next
open next
```

上記のどれでも同じ働きをします。「next」はカードナンバーが増えて行く方向に進みますが、逆向きに進めたい場合は

```
go to the previous card
```
前のカードに行きなさい

これも「next」と同じ言い換えができます。
また「previous」は短縮形の「prev」も使う事ができます。

```
go prev
```
（カードと言う言葉は省かれていますが、意味は）前のカードに行きなさい

カードにはそれぞれナンバーが付いているので、
カードナンバーを指定して移動する事もできます。

```
go to card 1
```
カードナンバー１に行きなさい

始めにメッセージ・ボックスからカード1を開いて、その後

```
go to cd 3
```
カードナンバー３に行きなさい

をメッセージ・ボックスから送って、カード3を開いてください。
「card」の短縮形「cd」を使ってみました。

ここでLiveCodeは一旦終了（Quit）させます。メッセージ・ボックスに

```
quit
```
終了しなさい

をメッセージ・ボックスから送って、この章は終わります。

Tips

- スタックやカードのように、別な呼び方や短縮形にできるオブジェクトは、呼び方を統一しておかないと、 何かの事情ですべてを書き直さなくてはいけない場合、検索から漏れてバグの原因となることがあります。

この章で出て来た言葉

create
コマンド（command）現在あるカード上に新しいオブジェクトを作成する
stack
オブジェクト（object）LiveCodeの開発の土台となるオブジェクト　同義語：window
card
オブジェクト（object）スタック内の１枚のカード　短縮形：cd
```
create stack "Hello World"
craete card
go to cd 1
```

set
コマンド（command）あるバリュー（価値、評価）を付与する、設定する
the
キーワード（keyword）プロパティやファンクションの名前の前に付けて指し示す
name
プロパティ（property）オブジェクトを特定する名前

this

キーワード（keyword）現在のスタック、カードを指し示す

```
set the name of this stack to "Hello World"
```

go

コマンド（command）　他のカードやスタックに移動する　同義語：open

next

キーワード（keyword）現在のカードから次のカードを指定する

previous

キーワード（keyword）現在のカードから前のカードを指定する　短縮形：prev

```
go to the next card
go next cd
go prev card
```

quit

コマンド（command）LiveCodeまたは作成したアプリケーションの終了

```
quit
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

03: オブジェクトとプロパティ

この章の概略

- *メニューからオブジェクト（スタックとカード）を作成して、インスペクタで名前等のプロパティ（オブジェクトの持っている様々な特性・属性）を設定します。*
- *スタックは「Name, Title, ID」の3種類の名前が付けられています。*
- *プロジェクト全体の構成をアプリケーション・ブラウザーから見ます。*
- *LiveCodeの書類を保存する際の注意。*

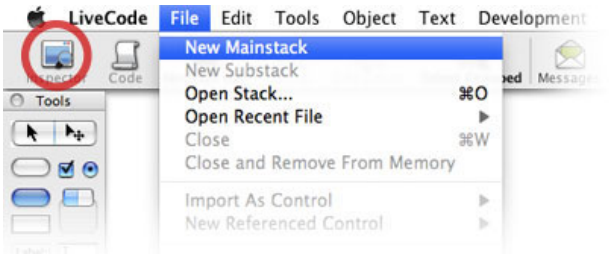
この章で使われる英単語

- property [名詞：（複数 properties），（一義的には）財産, 所有物, （モノの）特性, 属性]

メニューからスタックを作る

前章ではメッセージ・ボックスからLiveCodeの命令文を送って、スタックとカードとボタンの3種類のオブジェクトを作りました。LiveCodeはイラストレイタのようなグラフィック作成アプリではなく「プログラミング言語」ですから、すべてのオブジェクトの作成機能は、プログラミング言語のLiveCodeが持っています。しかし「統合開発環境」としての機能性から、直接メニューやツールを使ってオブジェクトを作成した方が効率的でしょう。初級のアプリ作成ではプログラミングでオブジェクトを作る事はあまりないですが、上級になるとオブジェクトや作成ツール自体を作るプログラミングが必要になる事も出てきます。この章で使うメニューやツールも、メニューやツールで作るオブジェクトも、作ったオブジェクトの属性（properties プロパティ）の設定も、プログラミングで作りだせる事を意識しておくと、プログラミング言語LiveCodeの理解が早まるかと思います。

今度はスタックをメニューから作ります。
「File」メニューから「New Mainstack」を選びます。（下図ファイル・メニュー）



「Untitled 1*」と言う名前のスタックがひとつ作られます。始めに作るスタックを「メイン・スタック」と言って、開発してゆく行くプロジェクトの中心になるスタックです。必要があれば、さらにその下に複数の「サブ・スタック」が加えられます。LiveCodeの作成するアプリは、必ずひとつのメイン・スタックと、メイン・スタックの中に 1 枚のカードが必要です。メイン・スタックを作った時点で、すでにカードは 1 枚作られています。

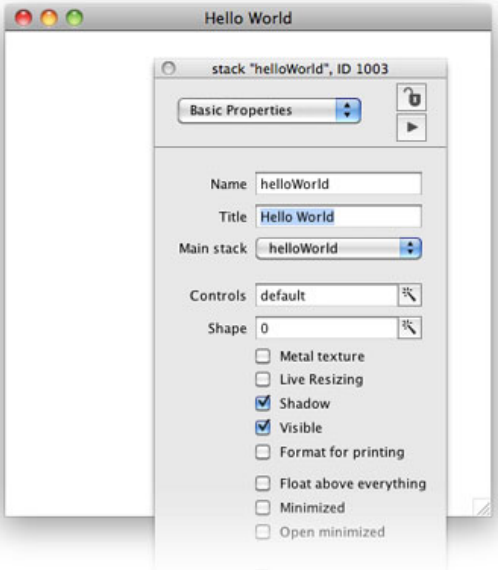
メイン・スタックはサブ・スタックとは違う特徴をいくつか持っていて、
プロジェクトの構成を考える際、それを考慮しなくてはいけませんが、

まだその事については考える必要はありません。

スタックのプロパティ設定

前章と同じようにスタックに名前を付けます。今回は開発環境のインスペクタを使います。メニュー・バーの下に並んだアイコンから「Inspecter（上図赤丸）」をクリックすると、スタック・インスペクタのパレットが現れます（下図）。インスペクタは、オブジェクトが持っている幾つものプロパティ（properties 特性、属性）を、確認したり編集したりするパレットです。スタック、カード、その他のオブジェクトの3種類のインスペクターがあり、今見ているのは「スタック・インスペクタ」で、このスタックの名前や大きさなど、普通よく使われるプロパティを編集する事ができます。

インスペクタからは設定できない、あまり使われないプロパティは、
LiveCodeのプログラミングで設定できます。またカスタムでプロパティを作ることもできます。



メニューから作ったばかりのスタック・インスペクタのトップにあるバーには「stack "Untitled 1", ID 1003」（IDの数字は違っているかもしれません）とあります。そのすぐ下のボタン・メニューには「Basic Properties」とあって、作られたばかりのスタックには、「Untitled 1」と言う名前（name）がデフォルトで入っています。スタック・インスペクタのこのページに見えているのは、沢山あるプロパティの最も基本となる設定だけです。最上部のボタン・メニュー「Basic Properties」をクリックスすると、その他のカラーやサイズ等のカテゴリーに分けられて、このスタックのプロパティが設定できます。今はこのページ「Basic Properties」にある、上のふたつ「Name」と「Title」だけの設定をします。Nameにワード・スペースなしで「helloWorld」とタイプして、カーソルをいれたままリターンキーを叩きます。オブジェクトの名前（name）に日本語は使えません、必ず英数字を使用します。スタック・ウインドウのトップのバーが「Untitled 1 *」」から「helloWorld *」に変わります。スタック・インスペクタの「Title」の欄には二つの単語（ワード・スペースを取って）で「Hello World」とタイプして、カーソルを入れたままリターンキーを叩きます。こんどはスタック・ウインドウのトップのバーが「Hello World」に設定されます。

スタックのをタイトルを日本語にする事もできますが、このスタック・インスペクタからはできません。
初歩的な日本語に関しては、後の章でまとめます。

スタック・プロパティの「name」と「title」の違い

上の経過から分かる事は、インスペクタの「Title」が空白の時には、スタック・ウインドウのトップ・バーは、インスペクタの「Name」に入れた「helloWorld」＋「* アスタリスク (asterisk)」が表示されました。特に書きませんでしたが、空白の「Title」の下にある「Main Stack」のボタン・メニューは、「helloWorld」に変わります。次に「Title」に「Hello World」を入れると、スタックのトップのバーは「Hello World」に設定されます。インスペクタの「Main Stack」のボタン・メニューは「helloWorld」のままで変わりません。

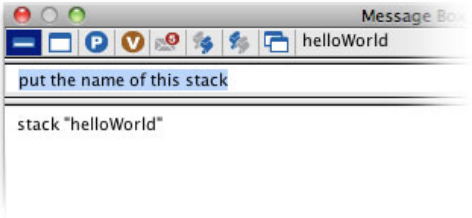
LiveCodeではひとつのスタック（オブジェクト）に、3種類のプロパティで名前を付けています。ひとつづつ説明をすると、まず「name」は、プログラミングの中で使うスタックの名前です。次に「title」は、スタック・ウインドウの最上部のバーに出すタイトルです。あるいは看板のようなモノと言っても良いでしょう。タイトル（title）は、アプリケーションを使用するユーザーが、そのウインドウの特性を理解しやすい名前でしたら、何ワードの語数でも数字でも設定できます。それに対してネーム（name）は、プログラミング中で使い安い簡潔な名前にします。「name」と「title」は必ずしも変える必要はありません。同じに「Hello World」としても差し支えないのですが、私の場合「name」はプログラミン中で使いやすく判断しやすいように、スペースは取り除いてその部分を大文字にし、ひとつの単語にしています。

基本的にスタックの名前（`name`）はダブル・クオートで括りますが
ワンワードの単語にしておくと、ダブル・クオートを外しても`name`として使う事ができます。
複雑に入り組んだスクリプト中では、ダブル・クオートなしの方が簡潔に書ける場合も時にあります。

もうひとつの名前の「ID」は、オブジェクトが作られた時にLiveCodeが自動的にそれぞれ違う数字（ユニークID）を割り当てます。スタック・インスペクタのタイトル・バーに表示されている「stack "helloWorld", ID 1003」の、カンマで区切られた2番目のアイテム（数字）です。今は「ID」と言うユニーク・ナンバーが割り振られている、と言う事を知っているだけにしておきます。

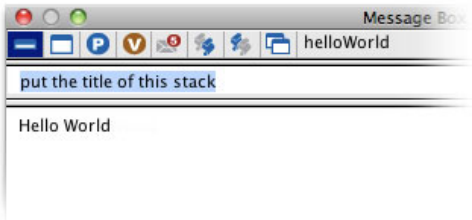
以上の事柄をメッセージ・ボックスから確認してみしょう。メニュー・バーの下のアイコン左から3番目「Message Box」をクリックするか、MacOSなら「コマンド・キー + M」、Windowsなら「コントロール・キー + M」のショートカットで、メッセージ・ボックスを開いて、以下の3つのスクリプトをメッセージ・ボックスから送ります。

```
put the name of this stack
```
このスタックの名前を書き出しなさい。



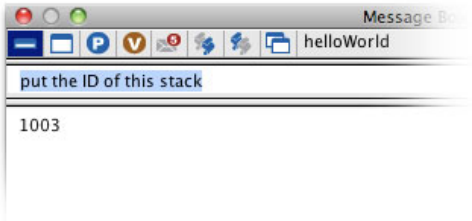

```
put the title of this stack
```

このスタックのタイトルを書き出しなさい。




```
put the ID of this stack
```

このスタックのIDを書き出しなさい。（LiveCodeはスクリプト中で大文字小文字の識別をしません）

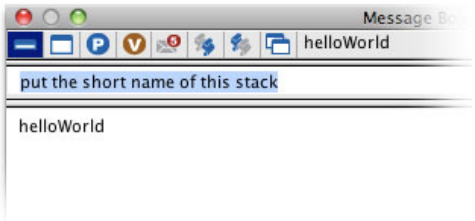


以上のような結果がメッセージ・ボックスに書き出されます。あるいは「the name of this stack」だけ予想された「helloWorld」とは違って、「stack "hellowWorld"」となったかもしれませんね。「the name of this stack」から返される値は、正式名称とでも言いましょうか「stack + ダブルクオートで囲われた名前」となります。期待する「helloWorld」だけ得たい場合は、状態を指し示すキーワード（keyword）の「short」を「name」に付けて

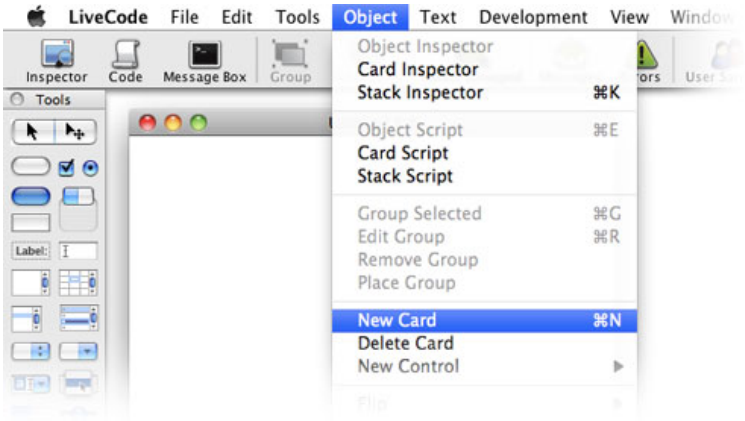
```
put the short name of this stack
```

このスタックのショート・ネームを書き出しなさい。

とメッセージ・ボックスから送ると「helloWorld」だけを得る事ができます。「stack 」を含む「the name」を使うか、「the short name」を使うかは、その時のプログラミングの流れの中で判断します。



メニューからカードを作る



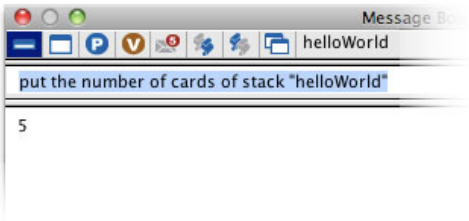
次にメニューを使って新しいカード・オブジェクトを作ります。「Objectメニュー」から「New Card」を選びます（上図）。スタックにはタイトルが設定されているので、前章の時のようにタイトルバーには「helloWorld (2)*」とカードの数字は表示されません。スタック中に何枚のカードがあるのか知るスクリプトを使います。メニューから数枚新しいカードを作ってください。始めにメッセージ・ボックスから

```
put the number of cards of stack "helloWorld"
```

スタック「helloWorld」のカードの総数を書き出しなさい。

```
put the num of cds of stack "helloWorld"
```

number とcards を簡略形にしても、まったく同じに働きます。

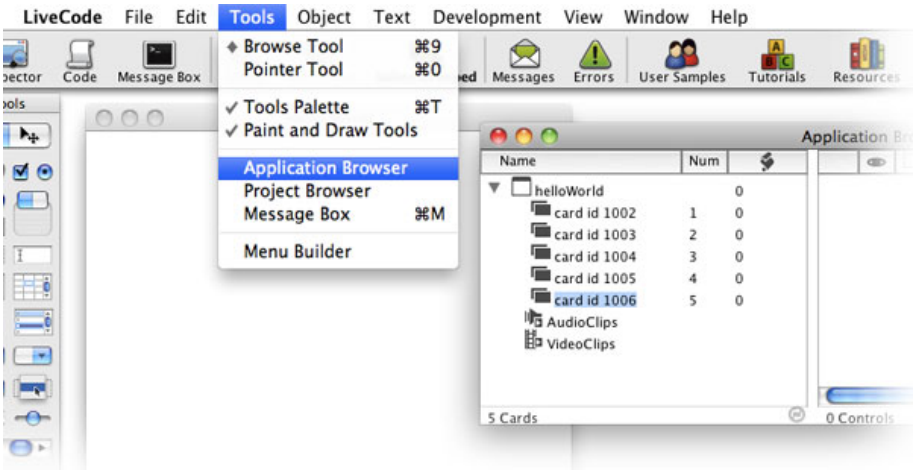


メッセージ・ボックスに、スタック「helloWorld」の中にあるカードの総数が返されます。LiveCodeは基本が英語ですから、「カード」は複数の「cards」にしないとエラーになります。その辺りが曖昧な日本語と違う発想に気をつけてください。と言ってもプログラミングで扱うオブジェクトですから、日常生活の語彙数とは格段に数が違うので、幾つかの言葉について気をつければ良いだけの話です。さらにカードの数を増やして、メッセージ・ボックスに書き出された数字が増えているか試してください。

アプリケーション・ブラウザーを使う

統合開発環境でオブジェクト等プロジェクト全体の構成を知るには、アプリケーション・ブラウザーを使います。（もうひとつLiveCode 6になって作られた、ほとんど同じ機能のプロジェクト・ブラウザーというのもありますが、カード・ナンバーが明記される等の理由で、ここではアプリケーション・ブラウザーを使う事にします）「Toolsメニュー」から「Application

Browser アプリケーション・ブラウザー」を開きます。



上図右にあるのが「アプリケーション・ブラウザー」です。スタック名「helloWorld」の左端の三角が横向きになってカードのリストが見えない時は、三角をクリックします。作ってある全てのカードのリストが現れ、「Name」の欄にはそれぞれのカードのIDナンバー、「Num」の欄にはカード・ナンバーが表示されています。その右の欄はオブジェクト内に書かれているスクリプトの行数で、「0」が縦に並んでいるのは、これらのオブジェクトには何もスクリプトも書かれていないのが分かります。

前章では指定したカードを開けるのに、メッセージ・ボックスからスプリプトで

```
go to cd 3
```

カードナンバー3に行きなさい。cdはcardの短縮形

のようにしてカードを開きましたが、アプリケーション・ブラウザーではカードのアイコンをセレクトしてから、ダブル・クリックします。実際にやってみましょう。ダブルクリックをしても、カード上には何もオブジェクトを作っていないので、果たしてそのカードか分かりませんね。それでは、メッセージ・ボックスから聞いてみます。

```
put the num of this cd
```

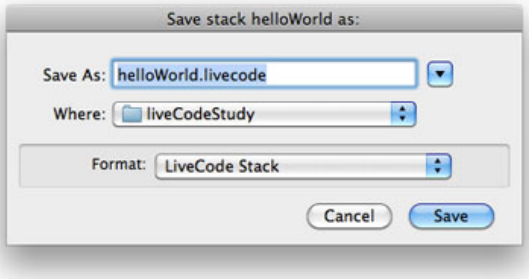
このカードのナンバーを書き出しなさい。numはnumberの短縮形

アプリケーション・ブラウザーで、ダブル・クリックしたカードの数字が返されましたか。

LiveCodeの書類を保存

ここでスタック「helloWorld」を保存します。LiveCodeは日本語を含むファイル・パスが使えません。具体的にはLiveCodeは日本語のファイル名は読めないので、必ず英語でファイル名をつけてください。もうひとつ日本語のフォルダー名も読めません。必ずフォルダーは英語名に

してください。私は「liveCodeStudy」と言うフォルダーに「helloWorld.livecode」と言う名前のファイルにしました。「Format:」はデフォルトの「LiveCode Stack」してください。他の「Format:」はLiveCodeの古いバージョンの書類形式で保存するためのものですから、使っているLiveCode 6 コミュニティ・エディションでは関係ありません。



もうひとつ、もし使っているOSを「拡張子 extension」が見えない設定にしていたら（拡張子は「helloWorld.livcode」のドットの後に来る「.livecode」等）拡張子が見える設定に変更してください。プログラミングをして行く過程で拡張子の判別が必要な事がしばしば出てきます。
Mac OSでは、
Finder メニュー > 環境設定 > 詳細 > すべてのファイル名拡張子を表示
Windowsでは（申し訳ないですが、英語版を使っているので日本語の表記がわかりません）
Control Panel > Appearance and Themas > Folder Options > View タブ > Advanced setting > Hide extensions for known file types のチェックマークを外します。

Tips

- LiveCodeはスクリプト中で、大文字小文字の区別をしません。区別の必要がある場合は、スクリプト中でその設定をします。それについて、ここでは記述しません。

- メッセージ・ボックスを開くショート・カット
  Mac OS: コマンド・キー + M
  Windows: コントロール・キー + M

- メッセージ・ボックスにタイプして行くと、今までメッセージ・ボックスから送った予測されるスクリプトが、グレーの文字で現れます。そのスクリプトをそっくり使いたい時は、キーボードの右矢印を叩きます。

- メッセージ・ボックスのスクリプトをタイプする所にカーソルを入れて、上下どちらかの矢印キーを叩くと、今まで使ったスクリプトが順番に現れます。

この章で出て来た言葉

`id`
プロパティ（property）オブジェクトに割り当てたユニーク・ナンバー
```
put the id of this stack
```

short

キーワード（keyword） 簡略形を指し示す

```
put the short name of this stack
```

number

プロパティ（property） オブジェクトの総数、または位置　短縮形：num

```
put the number of cards of stack "helloWorld"
put the number of this card
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

04: 測定と位置の設定

この章の概略

- *LiveCodeでプログラミングする長さの単位「ピクセル」*
- *スクリーンの大きさ、スクリーンの中心、その他のスクリーン内の測定*
  測定値のグローバルとローカルの違い
- *オブジェクトの横の長さ、縦の長さ*
- *マルチプル・メッセージ・ボックスの使用法*

この章で使われる英単語

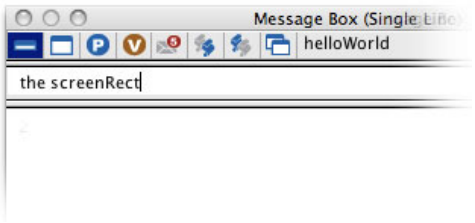
- rectangle [名詞：レクタングル, 矩形, 長四角形]
- location [名詞：ロケーション, 場所, 位置]
- item [名詞：アイテム,　項目, 事項]
- top [名詞：トップ,　最上部]
- bottom [名詞：ボトム,　最下部]
- left [形容詞：レフト,　左側の]
- right [形容詞：ライト,　右側の]
- width [名詞：ウィス, 幅, 横巾]
- height [名詞：ハイト,　縦, 高さ]
- global [形容詞：全世界の, 全体的な, 包括的な]
- local [形容詞：特定の場所, 局所的な, 狭い]
- ampersand [名詞：アンパーサンド, ＆記号]

スクリーンのサイズ

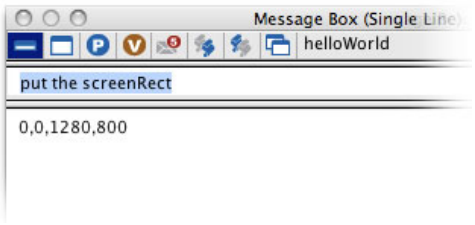
デスクトップ空間の大きさを探索してみます。アプリケーションの開発は、スタック（ウインドウ）と言う限られた四角形の内側だけでなく、デスクトップの隅から隅までの世界を使う事ができます。もちろんその先にはインターネットで繋がれた、もっと大きな宇宙があります。LiveCodeを終了させていたら、起動させてFileメニューから「Open Stack...」で、前章で作った「helloWorld.livecode」を開きます。保存していない時は、Fileメニューから「New Mainstack」を作ってください。 通常コンピュータの画面は長方形をしていて、ディスプレイとか、モニターとか、スクリーンとか呼ばれます。LiveCodeではコンピュータ画面を「screen スクリーン」と言う言葉で呼びます。スクリーンの大きさを求めるには、スクリーン（screen）とレクタングル（rectangle = 長方形）の合成語「screenRect（スクリーンレクト）」を使って、メッセージ・ボックスに

```
the screenRect
```

このスクリーンレクト（スクリーンの矩形）



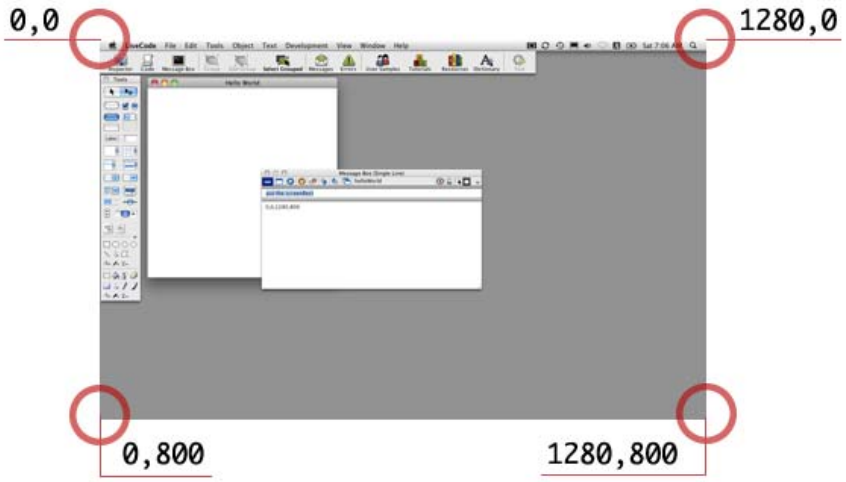
と、タイプしてリターン・キーを叩きます。「あれれれ？『put the screenRect into message box』じゃないの。」と思うでしょう。実はメッセージ・ボックスは、ほとんどの場合必要な「put」を省略できます。リターン・キーを叩くと「put 」が自動的に前に付いて、処理されて結果が返されます。




```
put the screenRect
```

（意味は）このスリーンの左上と右下のポイントのXYの値を4項目、メッセージボックスに入れなさい。

この場合も2章で説明した「set the name of this stack to "Hello World"」のように、特定のスクリーンを指し示す「the」が必要です。もし「the」を外して「screenRect」とすると、メッセージ・ボックス下段に文法の間違いで「error（エラー）」が表示されてしまいます。私は現在MacBookを使っているので、メッセージ・ボックス下段に「0,0,1280,800」がコンピュータから返されました。これはスクリーン画面の左上が、返された数字の項目１と項目２の「x=0, y=0」、スクリーン画面の右下が項目３と項目４の「x=1280, y=800」と言う、2カ所のピクセルの座標を表しています。この数字はそれぞれが使っているスクリーン・サイズで変わってきますから、他の数字が出ても問題ありません。



座標などと言うと、何やら難しそうな感じもしますが、コンピュータの画面は小さなピクセルの羅列で、センチメートルやミリメートルが普通の生活で使う長さの単位のように、LiveCodeではピクセルを長さの単位として使って行きます。

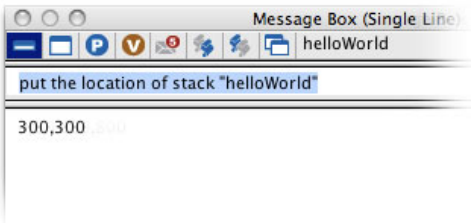
ピクセルは1インチが72ポイントと言う決まりの、昔からの印刷活字のサイズと同じ単位で
ポイントは現在でも、コンピュータのフォント（書体）の大きさで使われています。

メッセージ・ボックスに返された「0,0,1280,800」の項目1（item 1）と項目3（item 3）、つまりXの値を見てみると「x=0, x=1280」ですから、私の使っているスクリーンの横の長さは1280ピクセル、項目2（item 2）と項目4（item 4）つまりYの項目は「y=0, y=800」ですから、スクリーンの縦の長さは800ピクセルだと言う事がわかります。LiveCodeでは、メッセージ・ボックスに返された「0,0,1280,800」のようにカンマで区切られた文字列の項目は、「item 1」「item 2」「item 3」「item 4」のように「アイテム いくつ」と呼んでいます。「アイテム 3」はこの場合「1280」で、「アイテム 1」は「0」となります。

ではスタック「helloWorld」のこのスクリーンでの位置（location　ロケーション）を、メッ

セージ・ボックスから聞いてみます。LiveCodeでオブジェクトの「location ロケーション」と言った場合、オブジェクトの中心を指し示します。最初に「put」を付けても付けなくてもかまいません。

```
the location of stack "helloWorld"
```
スタック「helloWorld」のロケーションを返しなさい。



私の場合「300,300」がメッセージ・ボックスに返されました。人によって違う値が出るかと思います。これはアイテム１が座標X値の「300」、アイテム2が座標Y値の「300」ですから、スタック「helloWorld」の中心は「X=300, Y=300」の位置にあります。

それではスタック「helloWorld」をデスクトップの中心に移動させます。スクリーンレクト（screenRect）から、スクリーンの縦横のサイズが分かったので、それを２分の１にしてロケーション（location）を数字で設定する事もできますが、スクリーンに関する新しい言葉「screen　スクリーン」と「location　ロケーション」の合成語「screenLoc スクリーンロック」を使ってみます。始めにスクリーンの中心を確認してみましょう。メッセージ・ボックスに

```
put the screenLoc
```
このスクリーンの中心（スクリーンロック）を返しなさい。

リターン・キーを叩くと私のスクリーンでは「640,400」が返ってきました。「the screenLoc」を使って、スタック「Hello World」をデスクトップの中心に移動させます。

```
set the location of stack "helloWorld" to the screenLoc
```
スッタック「helloWorld」のロケーションをスクリーンロックに設定しなさい。

スッタック「helloWorld」が、デスクトップの中心に移動しました。確認してみます。

```
put the location of stack "helloWorld" into message box
```
スッタック「helloWorld」のロケーションをメッセージ・ボックスに入れなさい。

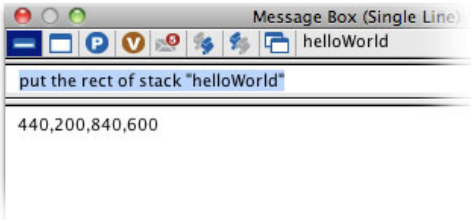
これで「the screenLoc」で得た値と、同じ数値が得られるはずです。

グローバル「global」と、ローカル「local」

「screenRect　スクリーンレクト」「screenLoc　スクリーンロック」「location ロケーション」の使い方は、だいたい分かった事と思います。では上記のバリエーションでスタック「helloWorld」のレクタングル（rectangle）を聞いてみます。メッセージ・ボックスに

```
put the rect of stack "helloWorld"
```

スッタック「helloWorld」のレクト（rectangle）をメッセージ・ボックスに入れなさい。

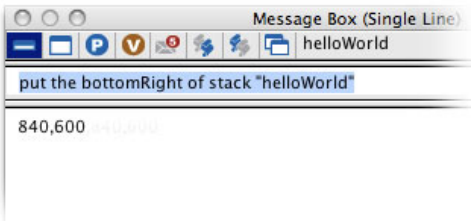


「rectangle」は短縮形の「rect」を使っています。カンマで区切られた4項目の数字が返されます。「the screenRect」と同じように、上図の始めの2 項目「440,200」はスタックの左上（topLeft　トップレフト）の座標、終わりの2項目「840,600」はスタックの右下（bottomRight　ボトムライト）のデスクトップ上の座標です。

```
put the topLeft of stack "helloWorld"
```

スッタック「helloWorld」のトップレフト（topLeft）を返しなさい。

課題：「topLeft」を bottomRight, top, left, bottom, right に入れ替えなさい。



図は課題の「the topLeft」を「the bottomRight」に入れ替えて得た結果です。「the topLeft」は「the rect of stack "helloWorld"」の始めの2項目（アイテム 1, アイテム 2）、「the bottomRight」は「the rect of stack "helloWorld"」の終わりのの2項目（アイテム 3, アイテム 4）と同じ結果を得る事ができます。

わざわざ「デスクトップ上の座標」と詳しく書いている意味は、LiveCodeの測定には「global　グローバル」と「local　ローカル」と言うふたつの基準があって、デスクトップ上の座標は「global　グローバル（デスクトップの左上が「0,0」）」、スタック内のオブジェクトの座標は「local　ローカル（スタックの左上が「0,0」）」として測定、位置設定をします。

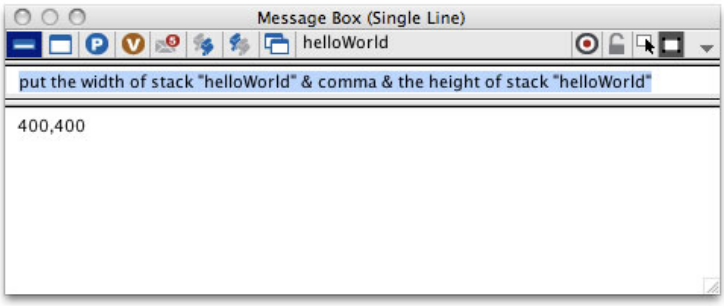
グローバルとローカルは座標だけでなく、スクリプトの中の値でも使います。後述。
上級では例外的に、スタック内の位置計測をグローバルで行うケースも出て来ます。

横の長さ「width」、縦の長さ「height」

スタックの横の長さも縦の長さも、上に書いた「the rect of stack "helloWorld"」を使って、引き算で求めることもできますが、直接オブジェクトを計測する「width ウィス（横巾）」と「height ハイト（縦、天地）」で求める事ができます。二つのアイテム（item）を繋ぐ「&　アンド記号（アンパーサンド）」と「comma カンマ（,）」を使って、メッセージ・ボックスに書き出します。

```
put the width of stack "helloWorld" & comma & the height of stack "helloWorld"
```

スタック「helloWorld」の横の長さ & スタック「helloWorld」の縦の長さをカンマで繋いで返しなさい。
カンマ記号をダブル・クオートでストリング扱いにして
put the width of stack "helloWorld" & "," & the height of stack "helloWorld"
と書いても同じです。



「400,400」がメッセージ・ボックスに出ます。つまり横巾が「400ピクセル（始めのアイテム）」天地が「400ピクセル（次のアイテム）」です。

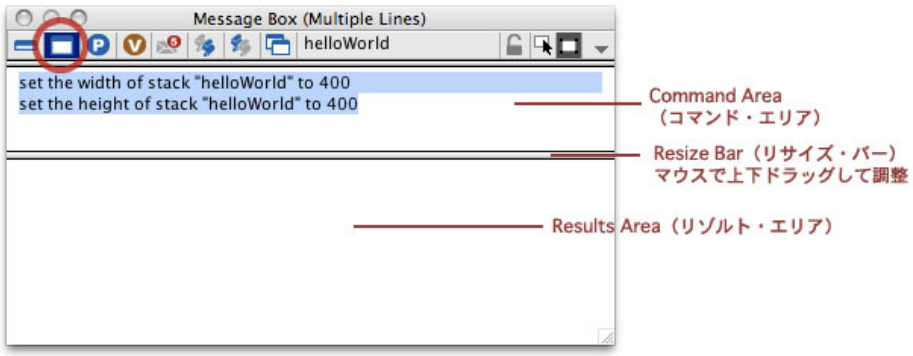
```
put the width of stack "helloWorld" & comma & the height of stack "helloWorld"
```

課題：スタック右下をマウスでドラッグしてスタックの大きさを変え、
メッセージボックスから上記スクリプトを送ってサイズを書き出す。

スタックの大きさを元に戻します。下図赤丸をクリックして、メッセージ・ボックスから複数行のコマンドが送れるマルチプル・ラインを使います。スクリプトを2行コマンド・エリアにタイプして、「エンター・キー」を叩きます。リターン・キーとエンター・キーがひとつになっているキーボードは、「コントロール・キー + リターン・キー」を叩きます。

```
set the width of stack "helloWorld" to 400
set the height of stack "helloWorld" to 400
```

スタック「helloWorld」の横巾を400に設定。
スタック「helloWorld」の天地を400に設定。



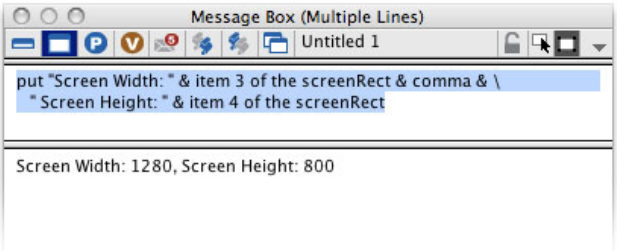
私たちには縦横サイズ同時に設定されたように見えますが、複数行のコマンドは上から1行づつわずかな時間差で処理されています。

「width ウィス」と「height　ハイト」は、オブジェクトの大きさを計る言葉で、スクリーン上の計測では使えません。上記で書いた「item　アイテム」の練習に、スクリーンの横と縦を求めてみましょう。「Screen Width: 」と「Screen Height: 」をストリングスとしてダブル・クオートで括る扱いにします。1行でも書けるスクリプトですが、長過ぎる場合は「\」（バックスラッシュ）を行替えする最後に付けて、その下に次の行として続きを書き続ければ、１行のス

テートメントとして扱われます。マルチプル・ラインのメッセージ・ボックスに

```
put "Screen Width: " & item 3 of the screenRect & comma & \
   " Screen Heighth: " & item 4 of the screenRect
```

**「スクリーンの横：」& スクリーンレクトのアイテム3と **
「スクリーンの縦：」& スクリーンレクトのアイテム４を、カンマで繋いで書き出しなさい。



タイプしたら「エンター・キー」または「コントロール・キー ＋ リターン・キー」を叩きます。

Tips

- スクリーンを2台コンピュータに繋いでいた場合「the screenRects」と複数形にすると、2行のスクリーンレクトが返されます。1行目がメイン・スクリーン、2行目のサブはメインの左上を0,0とした延長の数値になります。

- 「メッセージ・ボックスに返された」と言う言葉を、「メッセージ・ボックスに書き出す」と同じ意味で使っていますが、これは「Returning Values（コンピュータが結果としての値を返す）」の翻訳的な用語です。

- メッセージ・ボックスのシングル・ラインに打ち込んだコマンドは「リターン・キー」。メッセージ・ボックスのマルチプル・ラインに打ち込んだコマンドは「エンター・キー」、または「コントロール・キー + リターン・キー」を叩きます。

- LiveCodeの長さの測定は、デスクトップの左上が基準を「グローバル　global」、スタックの左上が基準を「ローカル　local」と分けて測定します。

- 行の長いステートメントは、1行分の見やすい処にバックスラッシュ(\)を入れて改行し、続きは次の行に書く事ができます。バックスラッシュを使った改行は何行でも可能です。ダブル・クオートで括ったストリングスの中で、バックスラッシュ(\)を使った行替えはできません。一旦ダブル・クオートで閉じて「&」で繋いでバックスラッシュ(\)を入れて改行し、ダブル・クオートで始めダブル・クオートで閉じます。
  例： "Alice was beginning to get very tired of sitting " & \
  "by her sister on the bank, and of having nothing to do."

この章で出て来た言葉

screenRect
ファンクション（function）スクリーンの左上右下2点のピクセル座標
```
put the screenRect
```

location
プロパティ（property）オブジェクトの中心位置を指し示す　短縮形：loc
```
put the location of stack "helloWorld"
```

screenLoc
ファンクション（function）スクリーンの中心位置を指し示す
```
set the location of stack "helloWorld" to the screenLoc
```

rectangle
プロパティ（property）オブジェクトのエリアを示す　短縮形：rec
```
put the rect of stack "helloWorld"
```

topLeft
プロパティ（property）オブジェクトの左上のXY座標を示す
```
put the topLeft of stack "helloWorld"
```

topRight
プロパティ（property）オブジェクトの右上のXY座標を示す
```
put the topRight of stack "helloWorld"
```

bottomRight
プロパティ（property）オブジェクトの右下のXY座標を示す　短縮形：botRight
```
put the bottomRight of stack "helloWorld"
```

bottomLeft
プロパティ（property）オブジェクトの左下のXY座標を示す　短縮形：botLeft
```
put the bottomLeft of stack "helloWorld"
```

top
プロパティ（property）オブジェクトの上辺のY座標を示す
```
put the top of stack "helloWorld"
```

Left
プロパティ（property）オブジェクトの左辺のX座標を示す
```
put the Left of stack "helloWorld"
```

bottom
プロパティ（property）オブジェクトの下辺のY座標を示す
```
put the bottom of stack "helloWorld"
```

right
プロパティ（property）オブジェクトの右辺のX座標を示す
```
put the right of stack "helloWorld"
```

width
プロパティ（property）オブジェクトの左端から右端までの長さ
```
set the width of stack "helloWorld" to 400
```

height

プロパティ（property）オブジェクトの上辺から下辺までの長さ

```
set the height of stack "helloWorld" to 400
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

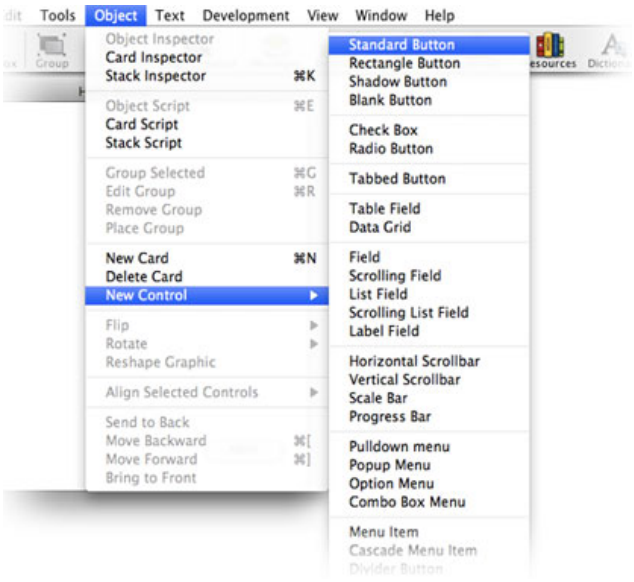
05: カード上のコントロール

この章の概略

- *カード上のオブジェクト「コントロール」とスクリプト・エディターの使い方。*
- *メッセージ・ハンドラー、エラーの扱い等スクリプトの初歩。*
- *コントロールをグループ化してバックグラウンドにする。グループの編集。*

コントロール「Controls」

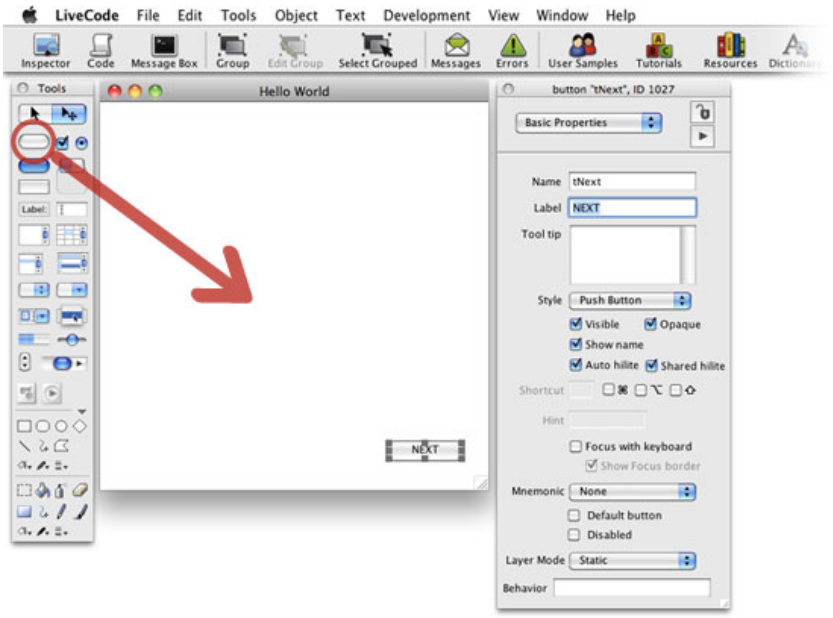
日本語で「コントロール」と言うと、「支配する」とか「制御する」と言う意味になりますが、LiveCodeでは、カード上に作られる全ての種類の「オブジェクト」を指します。全てのオブジェクトですから、ボタンか、テキスト・フィールドか、と言うオブジェクトの種類を問わず、作られた順にナンバーが振り当てられます。コントロールに付けられたナンバーは、上に重ねたレイヤーの階層順に付けて行くと考えてください。コントロールは「layer　レイヤー」と言う呼び方でも数えられます。二つ以上のコントロールをまとめたグループも、オブジェクトと考えるので、幾つかのオブジェクトを一つにしたグループも、コントロールの一つとしてカウントされます。グループはこの下で説明します。



第2章ではLiveCodeのスクリプトをメッセージ・ボックスから送って、ボタンをカード上に作りました。他にメニューバーから「Objectメニュー > New Control」を選んで、ボタン、テキスト・フィールド等、各種コントロールをカードの上に作ることができます（上図）。もうひとつはツール・パレットからドラッグ＆ドロップで、視覚的にオブジェクトをカード上に運んで作ります。たぶんツール・パレットを使うのが一番実用的かもしれません（下図）。

この章で使われる英単語

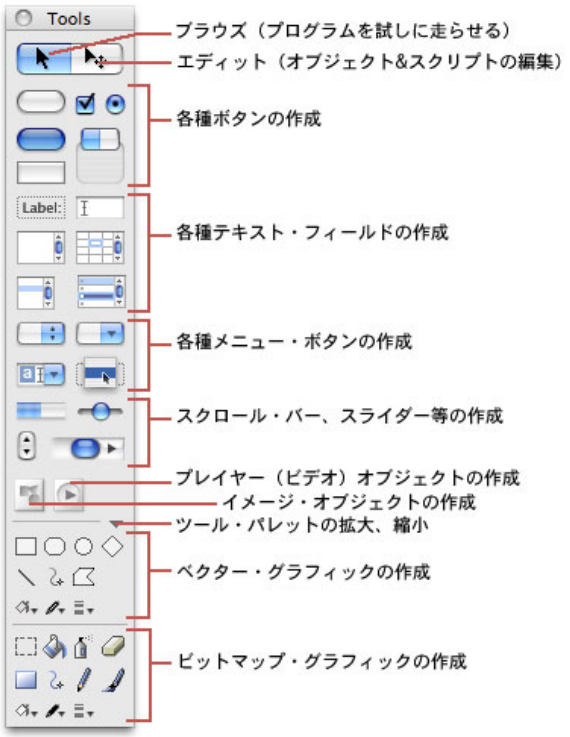
- control [名詞, 動詞：コントロール, 支配（する）, 制御（する）]
- layer [名詞：レイヤー, 層, 階層]
- browse [名詞、動詞：ブラウズ（する）, さっと目を通す（こと）]
- edit [名詞、動詞：編集（する）]
- editor [名詞：編集者, 編集用の機能]
- label [名詞：レイベル, ラベル]
- delete [動詞：削除する, 消去する]
- contextual [形容詞：前後関係の]
- message [名詞：メッセージ, 伝言, 情報処理上の単位]
- handler [名詞：ハンドラー, 扱い手, 情報を操作・処理する機能]
- apply [動詞：適用させる, 申し込む]
- group [名詞：グループ, ひとつに集めた物（事）]
- select [動詞：選ぶ]
- background [名詞：バックグラウンド, 土台, 背景]
- behavie [動詞：振る舞う, 行動する]
- behaivoir [名詞：振る舞い, 行動]
- share [動詞：シェアする, 共有する]
- place [動詞：置く, 配置する]
- field [名詞：フィールド, 野原, LiveCodeではテキストを表示するオブジェクト]



それでは前回からの延長で、スタック「helloWorld」を使います。スタックを開いてなかったら「Fileメニュー」の「Open Recent File」から「helloWorld.livecode」を選びます。「Tools Palette　ツール・パレット」が見えなかったら、「Toolsメニュー」の「Tools Palette」を選んでください。ツール・パレットから「Push Button　プッシュ・ボタン」（上図赤丸）をドラッグして、カード右下にドロップします。ドロップしたをボタンをダブル・クリックすると、そのボタンのプロパティ・インスペクタが現れます（または、オブジェクトが選ばれている状態で、上にあるアイコン・メニューの一番左「Inspector」をクリック）。インスペクタのNameには、デフォルトの名前「Button」があって、カード上に作ったオブジェクトのボタンには「Button」と表示されています。スタックの時のように「name　名前」を付けます。「tNext」と言うボタン名（name）にしました。インスペクタのName（名前）にボタン名をタイプした後、カーソルを入れたままリターン・キーを叩くと、オブジェクトに表示されるボタン名は「tNext」に変わります。スタックに「title　タイトル」と言うプロパティがあったように、ボタンにも同じような「label　レイベル」と言うプロパティがあります。インスペクタにあるLabel（レイベル）は大文字で「NEXT」とします。タイプしたら、リターン・キーを叩いてレイベルを設定すると、ボタンの表示は大文字の「NEXT」に変わります。スタックの場合と同じように、name（名前）はプログラミング中で使い、lable（レイベル）はアプリケーションを使うユーザーの為の表示です。今はボタン・プロパティの設定は、そのふたつだけとします。

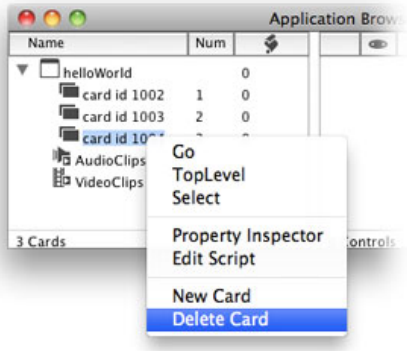
オブジェクトの名前（name）は、必ず変更しなくてはいけないと言うものではありません、
プログラミング中で使う必要がなかった、デフォルトのままでも良いです。
また「name」はスクリプト中では通常ダブル・クオートで囲うので、ワンワードにする必要はありません。

ツール・パレット「Tools Palette」



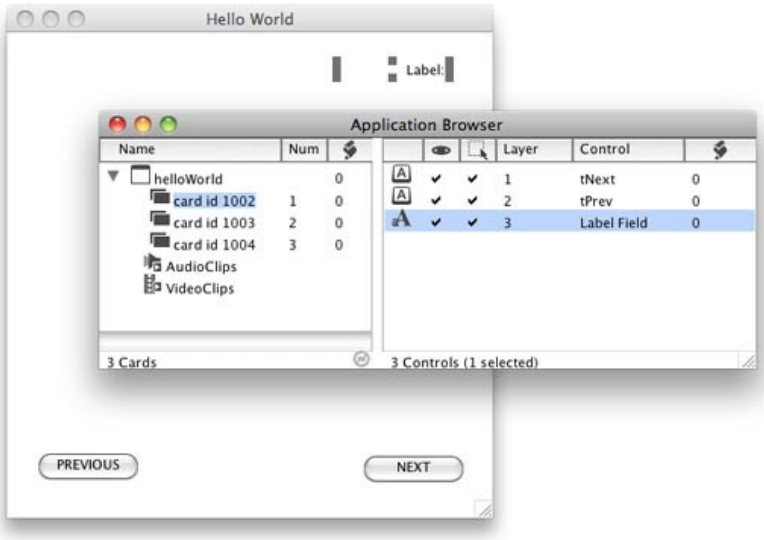
ツール・パレットから作ることができるオブジェクト（コントロール）は、上図の説明を見てください。いちばんトップにある二つの矢印は、左が「Browse Tool　ブラウズ・ツール」書いたスクリプトが正しく働くか、試しで動かしてみる時に使われます。その右の矢印に十字が付いているツールは「Edit Tool　エディット・ツール」または「Pointer Tool　ポインター・ツール」と呼ばれる、オブジェクトを選んでスクリプトを書き込んだり、オブジェクトの移動やサイズの変更に使うツールです。ツール・パレットからオブジェクトを作成すると、自動的にブラウズ・ツールはエディット・ツールに切り替わります。

次にカードの左下に同じく「Push Button　プッシュ・ボタン」を作ってください。やはりダブル・クリックでプロパティ・インスペクタを開き、名前（name）は「tPrev」、レイベル（label）は大文字で「PREVIOUS」とします。もう一つオブジェクトを作ります。ツール・パレットのテキスト・フィールドの中から左上にある「Label」を、カードの右上にドラッグ&ドロップします。テキスト・フィールドの「Label」はデフォルトのままで、特にプロパティを変更する必要はありません。このレイベル・フィールドはテキストの打ち込みがロックされていて、テキストは右寄せ（align right）、長いテキストが入っていても左右の幅で折り返されない設定になっています。「Toolsメニュー」から「Application Browser　アプリケーション・ブラウザー」を開きます。



アプリケーション・ブラウザーの左上スタック「helloWorld」が閉じられていたら、三角アイコンをクリックしてスタック内のカードが見えるようにします。スタックには最低１枚のカードがあります。ここでは3枚のカードにしましょう。もしそれ以上ある時は不要なカードをアプリケーション・ブラウザーの中でセレクトして、右クリックでコンテクストメニュー（contextual-menu）を開き「Delete Card」を選びます（またはコントロール・キーを押しながらマウスで左クリックすればコンテクストメニューが開きます）。「Delete Card」を選ぶと確認のダイアログが出ますから「Yes」をクリックします。カードが3枚ない時はやはりコンテクストメニューを開き、「New Card」で3枚にします。

「Delete Card」も「New Card」も、「Objectメニュー」からもできます。」

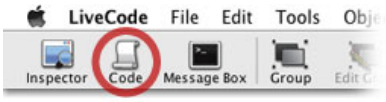


アプリケーション・ブラウザーのカードのトップ「card 1」（カード・ナンバー１）をセレクトしてください。図のようにセレクトしたカードがハイライトになって、右側にカード上のオブジェクトの一覧が見えます。今はカード・ナンバー１にしかオブジェクトを作っていません。上図ではカード上の「Label Field」が選ばれているので、オブジェクトのリストでも「Label Field」がハイライトになっています。反対にアプリケーション・ブラウザーでオブジェクトをハイライトにすると、カード上のオブジェクトも選ばれます。アプリケーション・ブラウザーの一番下を見てみると、左にカードの枚数（3 Cards）、右にコントロールの数（3 Controls (1 selected)）が表示されています。

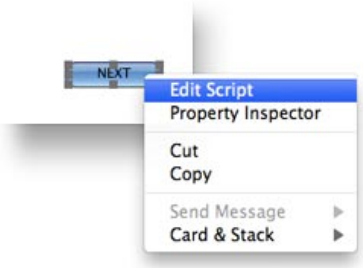
スクリプト・エディター「Script Editor」

LiveCodeでオブジェクトと呼んでいるモノは、すべて目に見える対象となっていて、その中にスクリプトを書き込むことができます。オブジェクトにマウスで指示を与えると、書かれてあるスクリプトはどういうマウスの指示だったかと判断してから、その後のスクリプトに従って（必要に応じて）、他のオブジェクトや、カード、スタックに指示を伝達して、目的とする処理が実行されます。ではオブジェクトの中にスクリプトを書いて行く事にしましょう。オブジェクトのスクリプト・エディタを開くには、幾つもの方法があります。今は知識としてして幾つもの開け方があると知っていれば良いですが、どれも状況に応じて使えるようになると便利です。

1）ツール・パレットでエディット・ツールを選んだら、スクリプトを書き込みたいオブジェクトをセレクトして、アイコン・メニューの左から2番目「Code」（下図）をクリックします。



2）スクリプトを書き込みたいオブジェクトをセレクトして、右クリックでコンテクストメニュー（contextual-menu）を開き「Edit Script」を選びます。使っているのがマウスを繋いでいないラップトップでした、コントロール・キーを押しながら、オブジェクトをクリックしてコンテクストメニューを開きます。



3）スクリプトを書き込みたいオブジェクトをセレクトして、

MacOSなら、コマンド・キー ＋ E

WIndowsなら、コントロール・キー ＋ E　でも開く事ができます。

これ以外にもプロパティ・エディターからも開けます。

メッセージ・ハンドラー「message handler」

それではボタン名「tNext」レイベル名「NEXT」のスクリプト・エディターを開いてください。ボタンのエディターにはあらかじめ「mouseUp」のメッセージ・ハンドラー（message handler）が用意されています。

```
on mouseUp

end mouseUp
```

メッセージ・ハンドラーは「on」で始まり、その後にメッセージ名（この場合は mouseUp）が

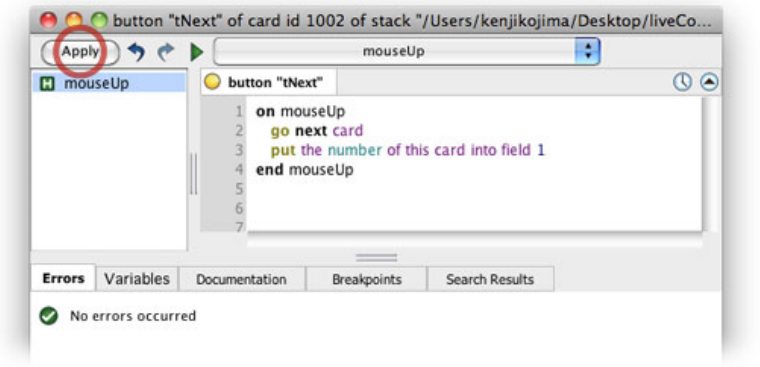
来て、さらに行を変えて必要なスクリプトのラインが入って（現在はまだ空白です）、最後に「end」と始めと同じメッセージ名「mouseUp」で終わります。この「mouseUp」のメッセージ・ハンドラーの意味は、ボタンをクリックして「mouseUp（マウスがアップ）」された直後に、ハンドラー内のスクリプトが上から順次コンピュータに送られ実行されて、行がすべて終わって「end mouseUp」に達した時点で、コンピュータの処理はハンドラーから抜けて終了します。マウスに関連するメッセージ・ハンドラーはこれ以外にも、「mouseDown（マウスがクリック・ダウンされた直後）」「mouseEnter（マウスがオブジェクト内の入った直後）」「mouseDoubleUp（マウスがダブル・クリックされた直後）」等、10以上のメッセージがあります。

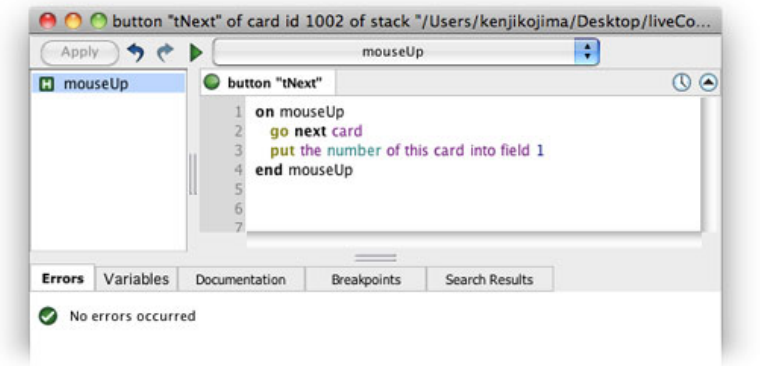
LiveCodeにはもう一つ重要な、ファンクション・ハンドラー（Function Handler）があります。

メッセージ・ハンドラーの中にスクリプトを書き込みます。

```
on mouseUp
  go next card
  put the number of this card into field 1
end mouseUp
```

次のカードに行きなさい
このカード（開いたカード）のナンバーをフィールド１に入れなさい



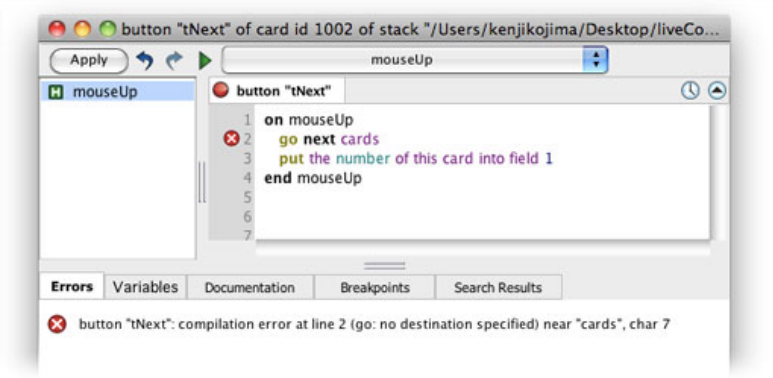


書き込んだスクリプトは、前の章でメッセージ・ボックスから送ったスクリプトと、ほとんど同じです。上の行はカードを移動させ、次の行は開いたカードのカード・ナンバーを、右上に置いたテキスト・フィールドに入れます。テキスト・フィールドは名前を付けていないので、フィールド・ナンバー「field 1」で指示をしています。

今カードには2つのボタンと、1つのフィールドがあります。

下からレイヤー階層の順序で（アプリケーション・ブラウザーの図では上から）
ボタンは「button 1」「button 2」、フィールドは「field 1」とナンバーが付けられています。
オブジェクトの種類別のナンバーは、インスペクタの「Size & Position」にあります。
オブジェクトのレイヤーの順序を変えると、ボタン・ナンバーやフィールド・ナンバーも変わります。

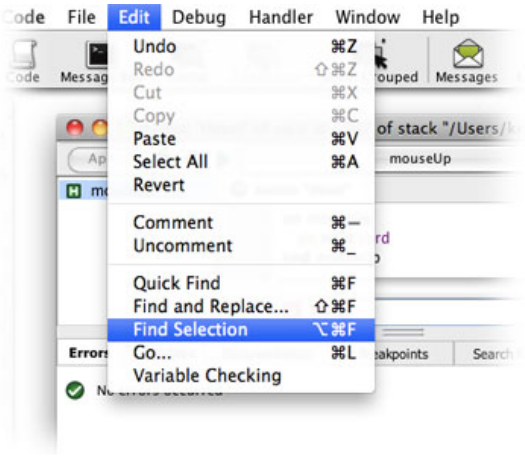
書き込んだスクリプトで、ボタン・オブジェクト「tNext」をクリック（マウスアップ）した時に、このスクリプトが働きます。ただし書き込んだだけではまだ働きません。スクリプト・エディター左上の「Apply」をクリックするか（上図1赤丸）、MacOSなら「コマンド・キー + リターン・キー」。Windowsは「コントロール・キー ＋ リターン・キー」を叩くと、「button "tNext"」の左にある黄丸が緑丸に変わって（上図2）、スクリプト・エディター内のスクリプトは実行可能になります。

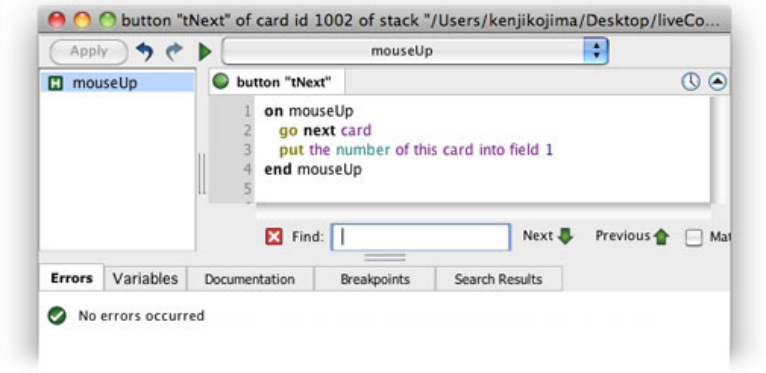


文法上の間違いやスペルの間違いがあると、「Apply」のクリックで、エラーが表示されます。スクリプトのどのラインがエラーなのか赤丸にバツ印で指示されます。上の図では「cards」と複数で書いてしまったので、「次のカード」が何なのかコンピュータに判断できなくなり、その行でストップされました。下にある「Errors」のタブに、その説明が英文で表示されます。これは開発環境に用意されている、コンピューターのプログラムを検査して、その誤り（バグ）を指摘する機能の「debug　デバグ」が働いたためです。エラーが出たら正しい文に直してもう一度「Apply」をクリックします。「button "tNext"」の左にある赤丸が緑丸に変わったらOKです。スクリプト・エディターを閉じます。

さらに追加したいスクリプトがある時等、エディターは開いたままでもスクリプトの実行はできます。
エディターを閉じるのは、通常のウインドウを閉じる機能を使う事も
MacOSなら「コマンド・キー ＋ リターン・キー」
Windowsは「コントロール・キー ＋ リターン・キー」を叩いてもできます

スクリプト・エディターを開いていたので、今後役に立つ「Find Selection」をエディター内に設置しておきます。スクリプト・エディターがアクティブになっているのを確認して「Editメニュー」から「Find Selection」を選んでください。Windowsではエクリプト・エディターのウインドウのトップにメニューがあります。





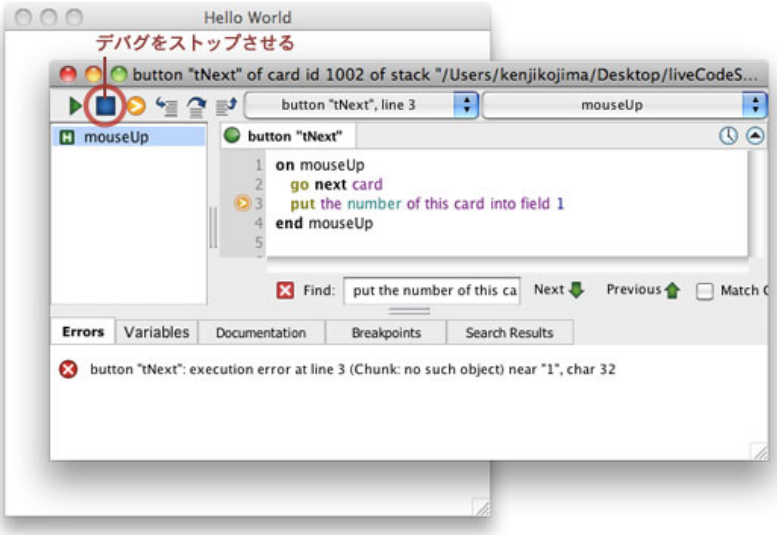
図のようにエディター内に「Find:」が現れます。スクリプトの行数が増えて、何か変更したり、ワードを探し出したりする時に役に立ちます。

今度は左下に置いたボタン名「tPrev」にスクリプトを書き込みます。ボタン名「tNext」と同じ手順でエディターを開いて

```
on mouseUp
  go prev card
  put the number of this card into field 1
end mouseUp
```

を書き込んで「Apply」でスクリプトを確定させてください。こちらはマウス・クリックで「前のカード」を開くスクリプトが１行目で、２行目はボタン名「tNext」とまったく同じです。

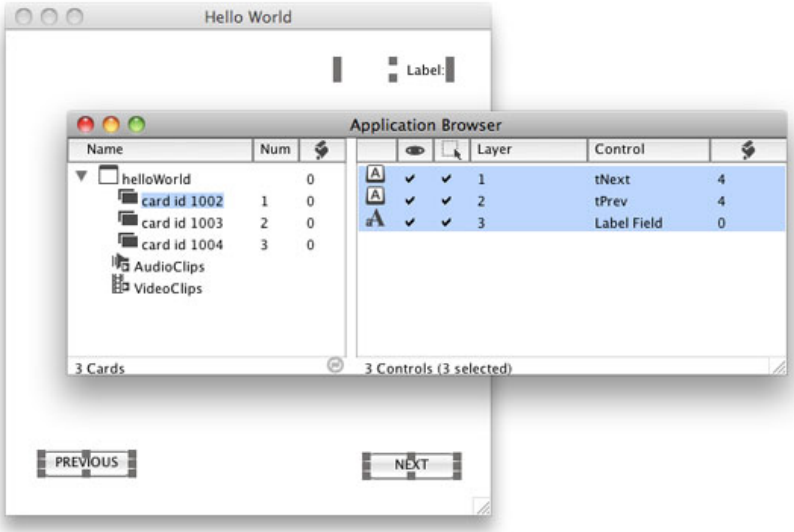
では早速試してみましょう。ブラウズ・ツールを選んで、右下の「NEXT」ボタンをクリックします。あららら。新しいカードが開かれた直後、ボタン「tNext」のスクリプト・エディターが開かれて、ストップしてしまいました。黄色の丸印でライン３を指し示しています。「put the number of this card into field 1」に問題ありですね。「カード2」を開いたのに、「field 1」があるのは「カード１」だけと言うのが原因です。いったんエディターを閉じてコントロールを修正することにします。スクリプト・エディターの左上に見える青い四角形（Stop Debugging）をクリックして、デバッギング状態を解除してからエディターを閉じてください。



コントロールをグループ化する

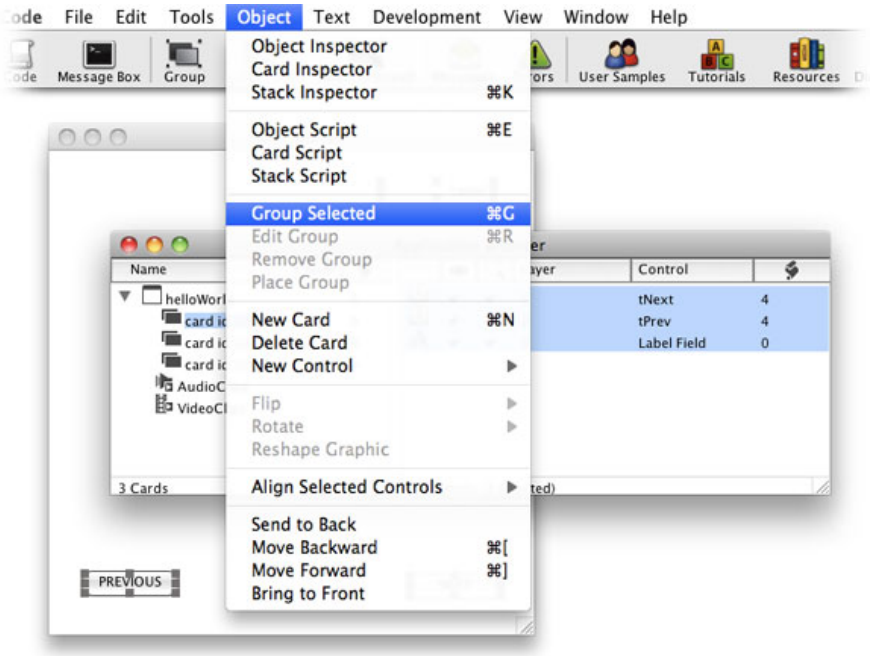
「カード2」を開いているので、ボタンもフィールドも見えません。「カード1」に戻ります。アプリケ–ション・ブラウザーの「カード1」をダブル・クリックしてください。「カード1」に戻ったので、3つのコントロールが見えています。ポインター・ツールで、カード上の3つのコントロールを選ぶと、アプリケ–ション・ブラウザーの3つのコントロールもハイライトになります。

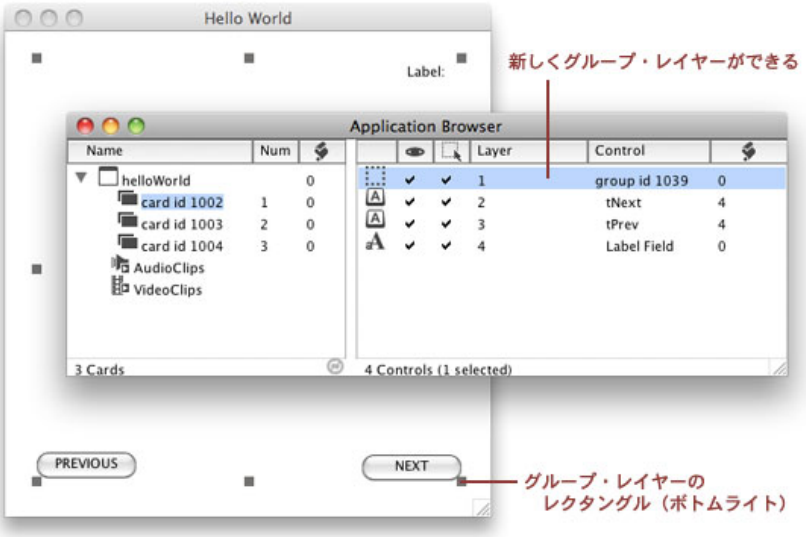
オブジェクトをまとめて選ぶのは、カード左上をクリックしたまま、マウスを右下に対角線にドラッグします。
または、Macではコントロール・キーを押しながら、Windowsではコントロールを押しながら
オブジェクトをひとつづつクリックして行きます。



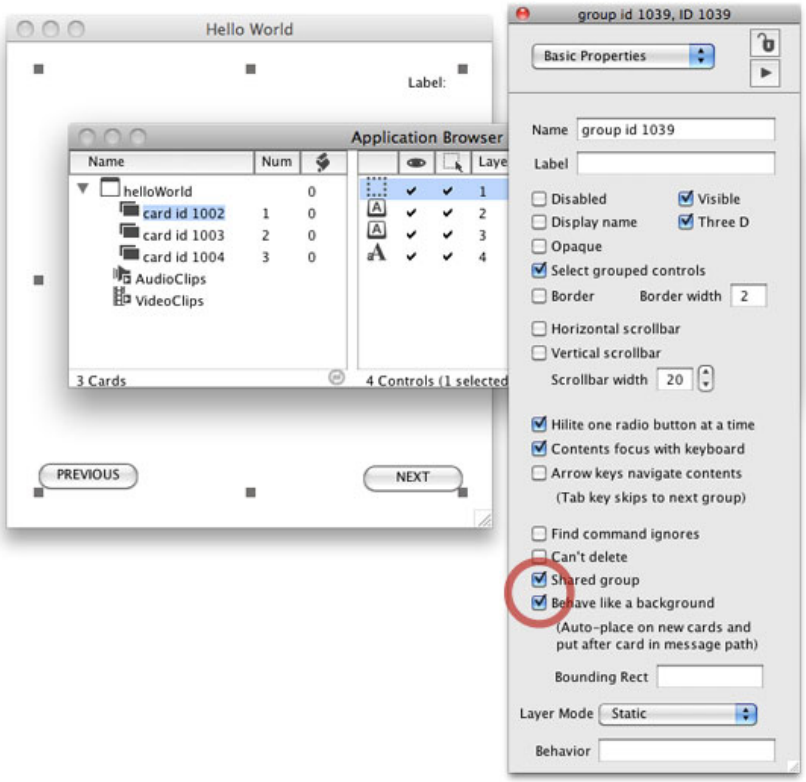
この状態で「Objectメニュー」から「Group Selected」を選ぶと、選ばれているオブジェクトはグループ化されて、一つのコントロールとして取り扱われるようになります。視覚的には透明なレクタングルの上に配置したオブジェクトのようになって、アプリケ–ション・ブラウザーの中にレイヤーで表示されます。グループもコントロールの一つですから、中にスクリプトを書

く事ができます（ここでは書きません）。



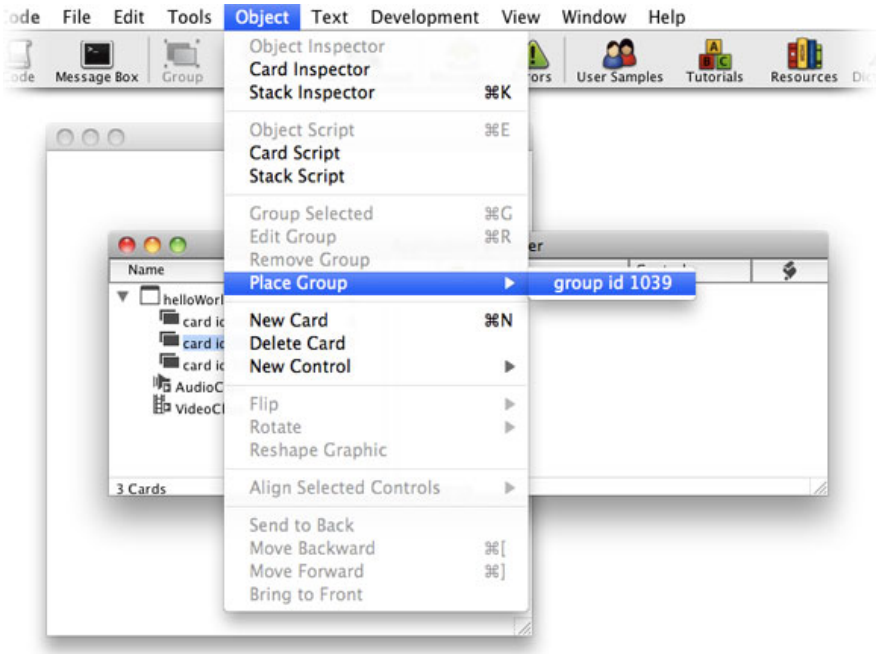


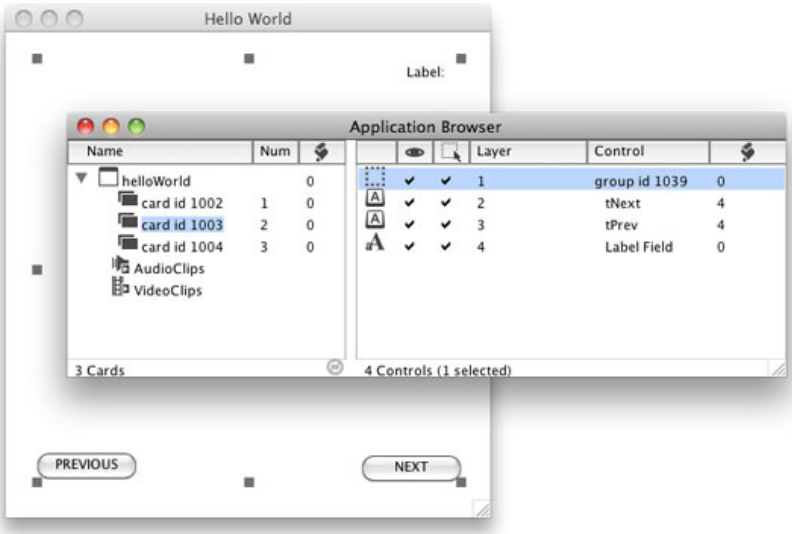
しかしこのままでは「カード1」にあるオブジェクトのグループですから、他のカードでも共通に扱われるようグループのプロパティを設定します。アプリケーション・ブラウザーのグループ「レイヤー1」をダブル・クリックすると、そのグループのインスペクタが出ますから「Behave like a background (backgroundBehaivoir)　バックグラウンドのような振る舞い」にチェック・マークを入れます。自動的に「Shared group」にもチェック・マークが入ります（下図赤丸）。



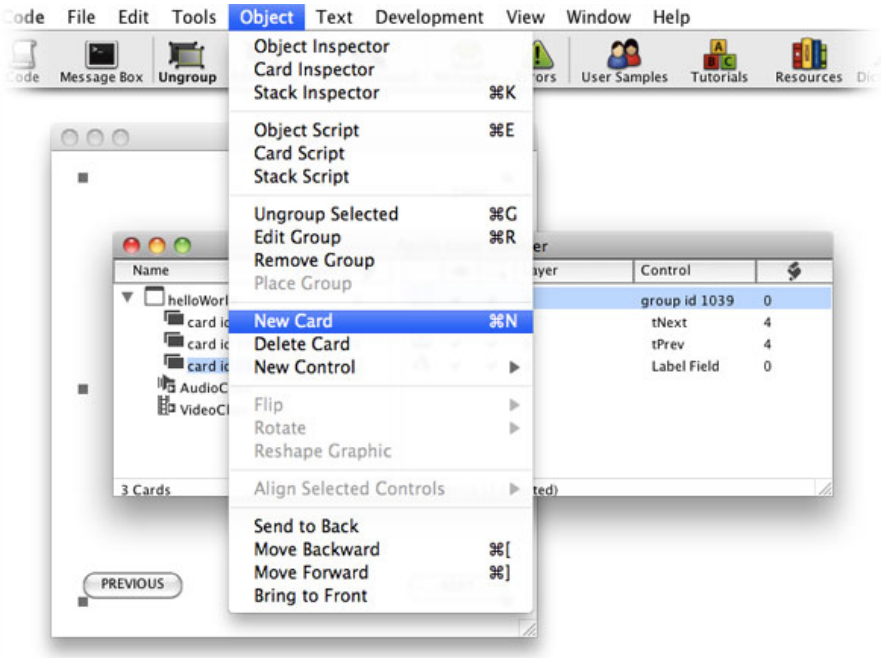
アプリケーション・ブラウザーの「カード2」をダブル・クリックして、「カード2」を開きます。カード上にはコントロールはありません。「Objectメニュー」から「Place Group」を選びグループを選ぶと、「カード2」にグループとその中にあるボタンとフィールドが現れます。

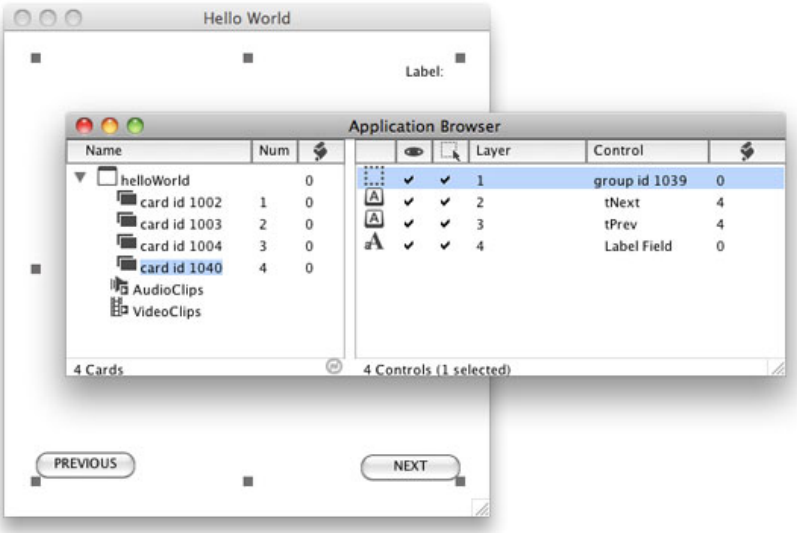
場合によってグループは一つだけとは限りません、「Place Group」に出て来るIDか名前を確認してください。





同じように「カード3」を開いて、「Objectメニュー」からグループを配置させます。「カード3」の次に、もう一枚新しいカード（カード4）を追加する事にしましょう（アプリケーション・ブラウザーで見ると一番下になります）。「Objectメニュー」から「New Card」を選びます。グループを作った時、プロパティを「Behave like a background (backgroundBehaivoir)　バックグラウンドのような振る舞い」に設定したので、カード上には共通のグループが自動で設置されます。

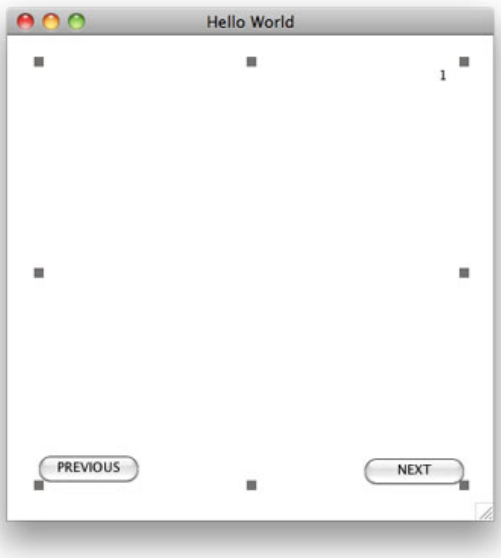


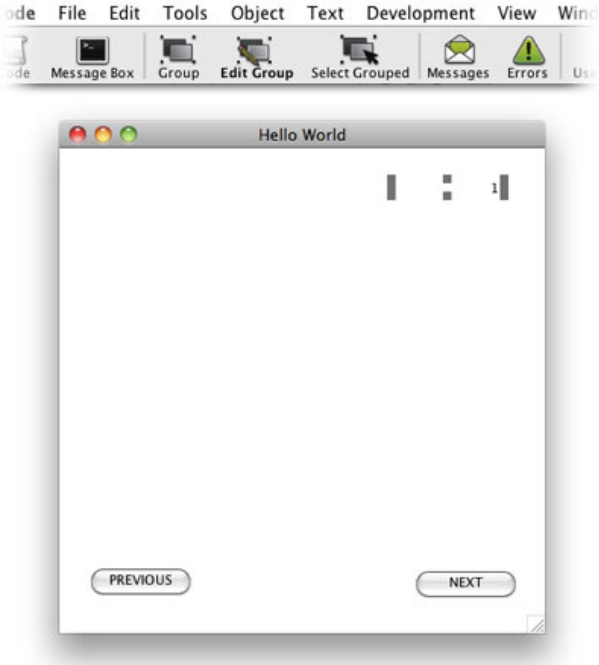


さあ、これでバグは修正されました。ブラウズ・ツールに持ち替えて「NEXT」ボタンをクリックして行ってください。カードが移動してそのカード・ナンバーが右上のテキスト・フィールドに表示されていますか。「PREVIOUS」ボタンの方はどうでしょう。

グループの編集

グループ化したボタンやフィールドのポジションを調整しようとして、ポインター・ツールで個々のオブジェクトをセレクトすると、グループ全体の透明なレクタングルが選ばれてしまいます。グループ内のポジション調整やスクリプト編集は、グループをセレクトしてから、アイコン・メニューの「Edit Group」（下図赤丸）をクリックすると、グループ化がいったん解除されて中のオブジェクトがセレクトできます。





ポジションを修正したら、もう一度アイコン・メニューの「Edit Group」をクリックすると、再びグループが現れます。もしオブジェクトの移動が始めのグループの透明レクタングルの外側になっていると、オブジェクトが見えなくなってしまうので、透明レクタングルにある四隅中央の小さな四角形を動かして、オブジェクトが見えるまでサイズ調整をしなくてはいけません。

最後にカード・ナンバーを「1」にしてから、スタックを「Fileメニュー > Save」して、この章を終わります。

Tips

- オブジェクトの名前（name）は、必ず変更しなくてはいけないと言うものではありません。ボタン・ナンバーやフィールド・ナンバーをスクリプト中で使う事もできますし、プログラミング中で使う必要がなかったらデフォルトのままでも良いです。

- スクリプト・エディターを開くには、
  ・オブジェクトを選んでアイコン・メニューから「Code」をクリック
  ・オブジェクトのコンテクストメニューから「Edit Script」を選びます。
  ・MacOSならオブジェクトを選んで「コマンド・キー ＋ E」
  ・Windowsならオブジェクトを選んで「コントロール・キー ＋ E」

- カード上のオブジェクトを、エディット・ツールで選んでダブルクリックすると、そのオブジェクトのプロパティ・インスペクタが現れます。 またはオブジェクトのコンテクストメニューから「Property Inspector」を選びます。

- スクリプト・エディターを開いたまま、ブラウズ・ツールでテストして、必要に応じてスクリプトを書き加えて行く事ができます。

- グループをバックグラウンドとして使うには、グループのプロパティ「「Behave like a background (backgroundBehaivoir)」を設定しなくてはいけません。

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

06: グラフィック、カラー

この章の概略

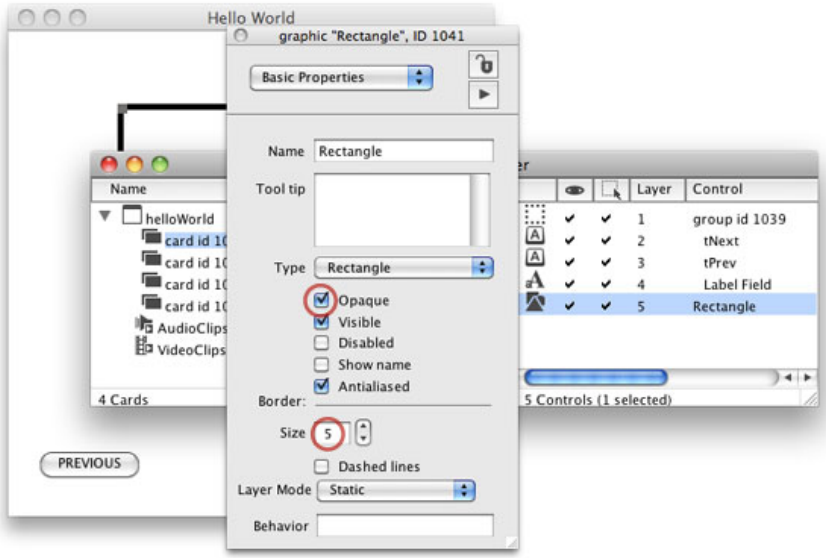
- *ツール・パレットからグラフィクス作成ツールで、レクタングル、ライン、フリーハンド・ポリゴンを作成します。*
- *グラフィク・オブジェクトのカラー・パレットを使った色彩設定や、RGBカラーと、普通の英語のカラー・ネームで、色彩設定のスクリプトを書きます。*
- *最後に練習問題があります。*

この章で使われる英単語

- opaque [形容詞：オペーク, 不透明な]
- border [名詞：ボーダー, 枠, 境界線]
- true [形容詞：真, 本当の]
- false [形容詞：偽, 誤った]
- lock [名詞：ロック, 錠]
- position [名詞：ポジション, 位置]
- foreground [名詞：フォーグラウンド, （風景絵画等の）前景]
- background [名詞：バックグラウンド, （風景絵画等の）背景]
- radius [名詞：半径]
- rounded [形容詞：丸みのある, 丸められた]
- Freehand [形容詞：自由に描いた]
- Polygon [名詞：ポリゴン, 多角形]
- reshape [名詞：リシェイプ, （新しい形に）作り直す]

レクタングルの作成

この章でも引き続きスタック「helloWorld.livecode」を使います。カード上にプロパティの異なるグラフィックを作って、カラーの設定等をして行きます。「カード1」を開いたら、ツール・パレットの「ベクター・グラフィック」作成から、左上の「Rectangle」を選んでカードの左上から右下にドラッグして、ほぼ正方形を描きます。パレットにベクター・グラフィック作成が見えない時は、パレット右下にある小さな三角をクリックして、パレットを伸ばせば現れます。グラフィックが描けたら、エディット・ツールに持ち替えてグラフィックをダブル・クリックするか、アイコン・メニューの「Inspector」をクリックして、グラフィック・インスペクタを開きます。



グラフィックのインスペクタを開いたページ「Basic Properties」にある「Opaque オペーク」にチェック、「Border　ボーダー」の「Size　サイズ」を「5」にしてください(上図赤丸2つ）。デフォルトではグラフィクの中が透明なので、オペーク（不透明）にして、カラーが表示できるようにします。デフォルトのオペークの色は白なので、見かけ上の変化は今見えません。ボーダーは5ポイントの太さにしました。デフォルトのボーダーの色は黒です。アプリケー

ション・ブラウザーを見ると、カード１には「Rectangle」と言う名前の新しいレイヤー「layer 5」ができています。この「layer 5」は「グラフィック・ナンバー」では「graphic 1」です。「グラフィック１」のプロパティの設定をメッセージ・ボックスから確認してみます。

```
put the opaque  of graphic 1
```
「グラフィック１」がオペークか返しなさい。

インスペクタでチェック・ボックスのあるプロパティは２者択一です。チェックした場合は「true　真」。チェックしてない場合は「false　偽」が返されます。この場合はオペーク（不透明）にしたので「true」が返されます。もしスクリプトでグラフィック１を透明に戻したい時は、「false」を使った下のスクリプトをメッセージ・ボックスから送ります。

```
set the opaque  of graphic 1 to false
```
「グラフィック１」のオペークを偽にしなさい（透明になります）。

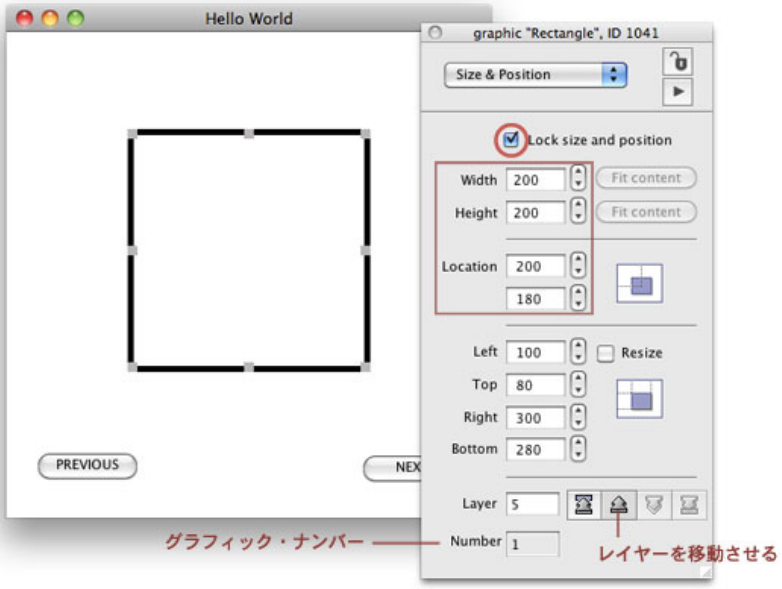
```
set the opaque  of graphic 1 to true
```
「グラフィック１」のオペークを真にしなさい（不透明に戻す場合）。

プロパティの真偽の変更をした場合、インスペクタは一旦閉じるか、ページを変えてもう一度開き直すかしないと、正しい表示がされません。次に「lineSize　ラインサイズ」を使って、ボーダー（border）のサイズを調べてみます。「lineSize」は、第４章にあった「screenRect」のように、「line」と「size」の二つの言葉が合わせられています。すべて小文字だけでも使えますが、LiveCodeを使うプログラマーは一般的に、読みやすく理解しやすいように、語の接続している文字を大文字にする事が多いです。ラインサイズを求める時に「line size」と２語にするとエラーになります。

```
put the lineSize of graphic 1
```
「グラフィック１」のラインサイズを返しなさい（「5」がメッセージ・ボックスに返されます）。

```
set the lineSize of graphic 1 to 10
```
「グラフィック１」のラインサイズを10にしなさい（ラインサイズを変更するスクリプト）。
ボーダーをなくしたい時はラインサイズを0にします。

サイズ、ポジションの設定



グラフィックのサイズと位置の調整をします。グラフィック・インスペクタの最上部にあるボタン・メニューから「Size & Position」を選んでください（始めにインスペクタを開いた時は「Basic Properties」になっています）。上図のようなページが開かれます。上から「Lock size and position　サイズと位置を固定する」にチェック・マークを入れます。これをしておくと、エディット・ツールでは動かなくなります（スクリプトでは動かす事ができます）。
「lockLocation」は「lock　錠」と「Location　位置」の合成語で、「ture　真」または「false　偽」で設定するプロパティです。

```
set the lockLocation of graphic 1 to true
```
「グラフィック１」のロックロケーション（lockLocation）を「真」と設定しなさい。

```
set the lockLoc of graphic 1 to false
```
「グラフィック１」のロックロケーション（短縮語：lockLoc）を「偽」と設定しなさい。

この二つのスクリプトで、グラフィックのロックまたはアンロックが切り替わります。グラフィック（レクタングル）の「width　横巾」を「200ピクセル」、「height　高さ」も「200ピクセル」にしましょう。「location　ロケーション（グラフィックの中心）」は、左から「200ピクセル（X値）」、上から「180ピクセル（Y値）」とします。Y値はグラフィックの下にある空間ではなく、カードの上辺からなので、間違えないようにしてください。カード内に置かれたオブジェクトのロケーションは、第２章で書いたようにローカル（local）ですから、カードの左上「0,0」が基準となります。なおメニューから作った、このデフォルトのスタック、カードの大きさは、400x400です。

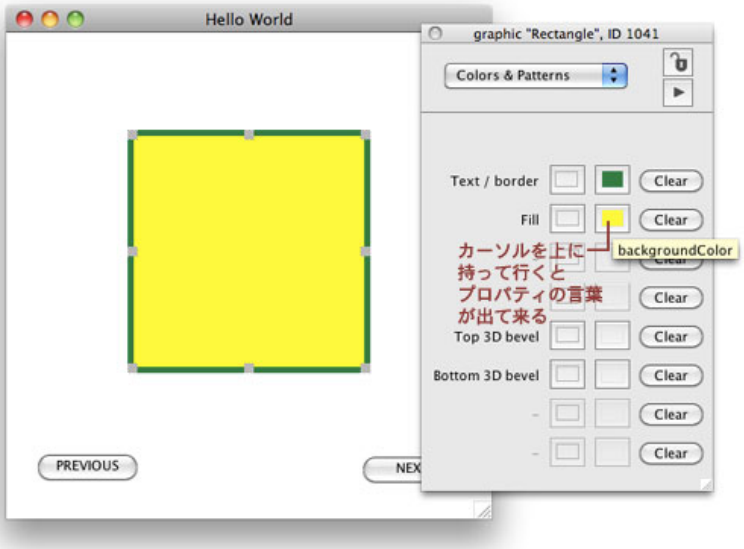
インスペクタの下部にある「Layer　レイヤー」は、このグラフィックのカード上のコントロールのレイヤー・ナンバーです。今は「layer 5」です。「Layer」の右にある４つのアイコンは、このグラフィックのレイヤーの移動です。左から「このグラフィックを一番下に移動させる。つまり「layer 1」にする」。次が「一つだけ下に移動させる」。数の上で考えると「layer 4」になりますが、今カード上にはコントロールを３つ含んでいるグループがこのグラフィックの下にありますから、グループになっているコントロールをまとめて飛び越して、このグラフィックのレイヤーは「layer 1」になります。次は「上に一つだけ移動させる（現在最上位にあるので、アイコンは薄くなってこの機能は使えません）」一番右は「最上位に移動させる

（現在最上位にあるので、アイコンは薄くなってこの機能は使えません）」。インスペクタの一番下「Number　ナンバー」はこのグラフィックのナンバーです。もしもうひとつこの上にグラフィックが作られると、新しいグラフィックのナンバーは「2」になります。

課題：インスペクタを使って「グラフィック1」のレイヤーを移動させ、元にもどす。
同時にアプリケーション・ブラウザーでもレイヤーの位置を確認する。

カラーの設定

グラフィックのカラーを設定します。インスペクタのトップにあるボタン・メニューから「Colors & Patterns」を選びます。「Text / border　テキスト／ボーダー」の右にある２つの四角形の右側をクリックしてください。カラー・パレットが現れます。任意のグリーンを選んで「OK」してください。グラフィックのボーダーが黒からグリーンに変わります。次に2列目の「Fill　フィル」の四角形、やはり右側をクリックします。同じようにカラー・パレットが現れます。任意の黄色を選んで「OK」してパレットを閉じます。グラフィックのボーダーの内側が黄色になります。もしカードにあるグラフィックの色が白いままだとしたら、始めの「ベーシック・プロパティ　Basic Property」のページで、「オペーク　Opaque」にチェックを入れてください。ここまでの図が下にあります。



上図のインスペクタに赤文字で書いたように、四角形の上にカーソルを当てると、小さな黄色の四角の中に言葉が現れます。LiveCodeではこれを「toolTip　ツールチップ」と呼んでいます。インスペクタの上の四角（今グリーンになっている）にカーソルを持って行くと、「foregroundColor　フォーグラウンドカラー」と言うツールチップが出ます。これはグラフィックの「foregroundColor　フォーグラウンドカラー」と言うプロパティに、この四角の色を設定したと言う事です。同じようにその下の「Fill　フィル」の右側の四角形にカーソルを持って行くと、「backgroundColor　バックグラウンドカラー」とツールチップが出ます。グラフィックのプロパティ「backgroundColor　バックグラウンドカラー」に、四角の色を設定しています。どういうカラーの設定をしたのか、メッセージ・ボックスから調べて行きましょう。

```
put the foregroundColor of graphic 1
```

グラフィック１のフォーグラウンド・カラー（foregroundColor）を返しなさい。

（フォーグラウンド・カラーはこの場合ボーダーです）

私の場合はMacOSのカラー・パレットから「Moss」と言う色彩名を選んだので、「0,128,64」が返されました。WindowsやMacOSでも他のをグリーンを選んだ人は、別な数字が返されます。これは普通コンピュータのカラー設定が、光の三原色の「Red（赤）, Green（緑）, Blue（青）」で決められているので、その数値が返されています。通常これを「RGBカラー」と言って、それぞれの色の数値は「0 〜 255」までの値を取ります。上で返された3つのアイテムで言うと「赤は0」「緑は128」「青は64」と言う数値です。厳密な色彩指定をするには、この方法が今の処一番良い方法です。

LiveCodeではもうひとつ簡易的な方法、日常的な英語に近い表現「colorNames　カラー・ネーム」も用意されています。

```
put the colorNames
```

（全部の）カラー・ネームを書き出しなさい。

100以上の色彩名が書き出されます。あまり沢山あり過ぎて何が何か分からないですね。ここに「LiveCode のカラー・サンプル http://kenjikojima.com/livecode/colorNames/」をインターネット・ブラウザーで見れるように作りました。インターネット・ブラウザーのカラー名で色彩が表示されるので、LiveCodeで設定した色名とは多少違いがあるのを了解してください（実際の色彩はLiveCodeで確認）。
例えば、このように使えます。

```
set the foregroundColor of graphic 1 to Blue
```

グラフィック１のフォーグラウンド・カラー（foregroundColor）をブルーにしなさい。

こう設定した後

```
put the foregroundColor of graphic 1
```

グラフィック１のフォーグラウンド・カラー（foregroundColor）を返しなさい。

これでプロパティのフォーグラウンド・カラー（foregroundColor）に、「Blue」と言う言葉が設定されたのがわかります。（レクタングルのフォーグラウンド・カラーは、四角形外枠の線です）

グラフィックのバックグラウンドカラー（backgroundColor = 四角形の内部）も、フォーグラウンド・カラー（foregroundColor = 四角形の外枠の線）と同じようにスクリプトで設定できます。

```
put the backgroundColor of graphic 1
set the backgroundColor of graphic 1 to 255,255,0
set the backgroundColor of graphic 1 to Yellow
```

グラフィック１のバックグラウンド・カラーを返しなさい。
グラフィック１のバックグラウンド・カラーを 255,255,0 に設定しなさい
グラフィック１のバックグラウンド・カラーを Yellow（黄色）に設定しなさい

RGBカラーの基礎知識

RGBカラーは光の3原色の「R (red)　赤」「G (green)　緑」「B (blue)　青」の「0 から 255 まで」の数字の組み合わせで色彩を作り出す、カラースクリーンのピクセルのシステムから、LiveCodeの色彩設定でも使われています。実際にはひとつのピクセルの中には、透明度を設定するアルファ・チャンネルか、画像のマスクが指定できるもうひとつのアイテムがありますが、通常の色彩設定では「RGB」の3つのアイテムを使用します。「0 から 255」と言うのは、数にすると0を含んで数えるので「256段階」あります。大雑把に言うと３つのアイテムの数字が「255」に近づくにしたがって白になり、「0」に近づくにしたがって黒になります。3つのアイテムを同じ数字にすると、グレーの色調が得られます。普通の絵の具の色彩混合とは感覚的に少し違うので、RGBカラーの設定をする時はスクリプトで試してカラーを見るのが良いでしょう。

下のスクリプトのように、RGBの数値を変えて色々試してみましょう。

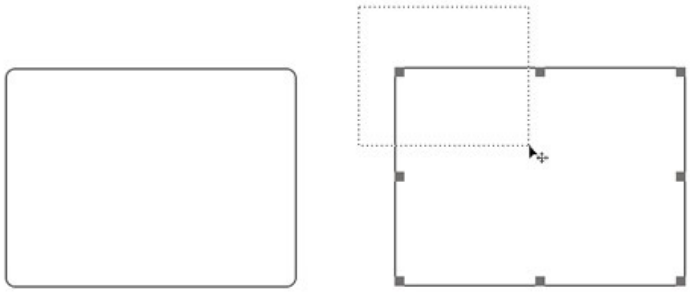
```
set the backgroundColor of graphic 1 to 255,255,0
set the backgroundColor of graphic 1 to 0,180,160
set the backgroundColor of graphic 1 to 100,100,100
```

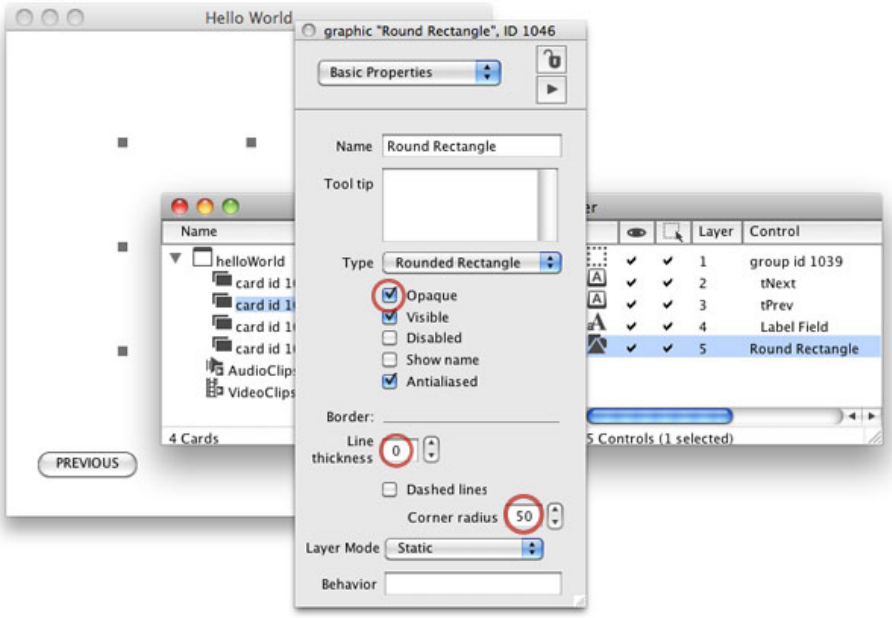
もうひとつWebのhtml原稿等で使う16進数のカラー表記も、
フォーグラウンド・カラー、バックグラウンド・カラーの設定に使うことができます。

```
set the foregroundColor of graphic 1 to "#FF00FF"
```

その他のグラフィックの作成

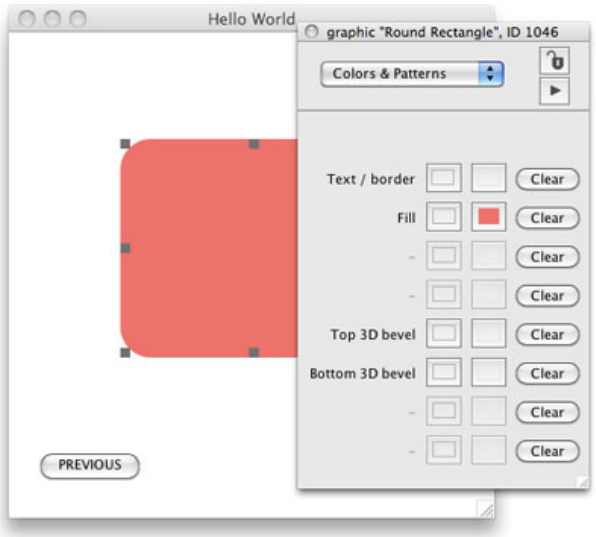
少し横道に入ったので、またスタック「helloWorld」に戻ります。ブラウズ・ツールでボタン「NEXT」をクリックして、カード２を開いてください。カード1と同じようにツール・パレットから今度はRectangleの隣の「Rounded Rectangle　ラウンド・レクタングル（丸みのある四角形）」を選んだら、カード２の上に左斜め上から右下に向かってマウスをドラッグして、少し横長の四角形を描きます。カード上には角のある四角形の枠が描かれますから、エディト・ツールに持ち替えると角に丸みのある「roundRectangle　ラウンド・レクタングル」になります。プロパティの設定をします。デフォルトのレクタングルは中が透明なので、一旦セレクトを外すと、マウスで四角形の中をクリックしてもオブジェクトをセレクトできません。外枠の細い線の上をクリックするか、下図右のように目的のオブジェクトをエディット・ツールで対角線に選んでセレクトします。セレクトした「ラウンド・レクタングル」をダブル・クリックするか、アイコン・メニューの「Inspector」のクリックして、グラフィック・インスペクタを開きます。





この「ラウンド・レクタングル」も「Opaque　オペーク（不透明）」にします。「Line thickness　ラインの太さ」は「0」です。ラインをゼロにすると、図のようにオブジェクトを囲っている8カ所の黒い四角だけが見えます。「Corner radius　角の丸みの半径」は「50」にしました。

インスペクタの「Colors & Patterns」を開きます。ラインをゼロにしているので、「Text / border」の「foregroundColor」は何も選びません。「Fill」はカード 1 とは違う任意の色を設定してください。私は「backgroundColor」に Salmon をカラー・パレットから選びました。外枠がなくて単一の色彩の、丸みのある四角形が現れます。

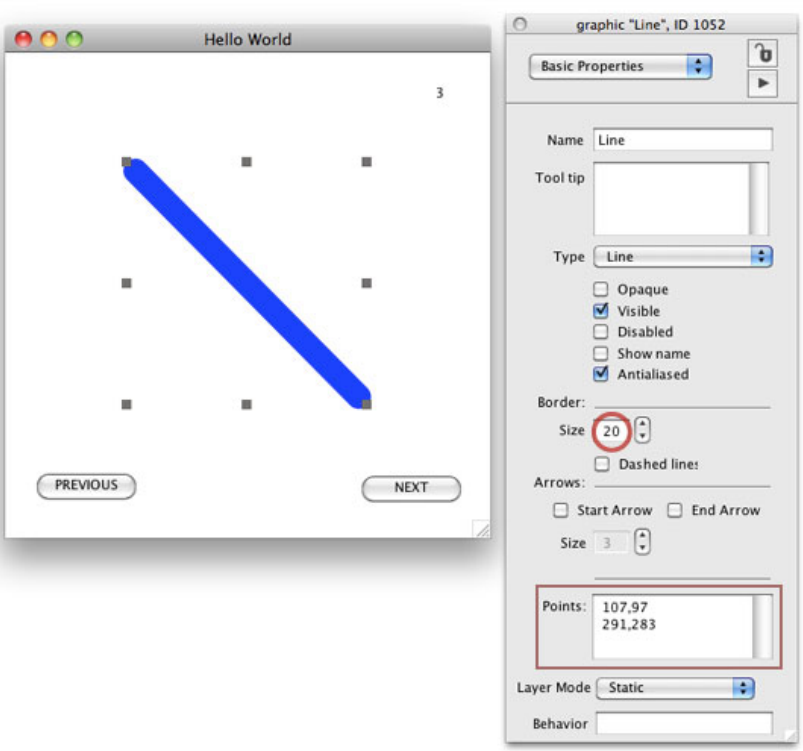


プロパティ・インスペクタ最初のページにある「Type」のボタン・メニューに、「Rounded Rectangle」とありますが、この型を持ったグラフィックのタイプ（Type）をスクリプトで知るには、「style　スタイル」と言う言葉を使います。

```
put the style of graphic 1
```
グラフィック１のスタイルを返しなさい（「roundRect」が返されます）

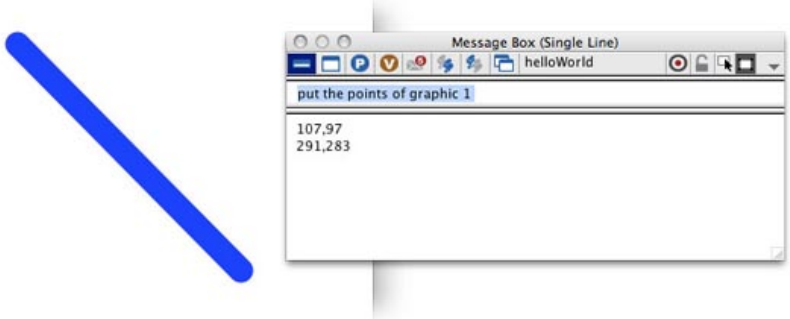
```
set the style of graphic 1 to roundRect
```
グラフィック１のスタイルをラウンドレクトに設定しなさい。

カード3には「Line　ライン」を描きます。ツール・パレットのベクター・グラフィック作成から「Line　ライン」を選んでカードに任意のラインを引いてください。オブジェクトのダブル・クリックでインスペクタを開いたら、「Border Size」を「20」にします。ラインのプロパティ設定はこれだけにします。インスペクタの下の方に「Points」と言うのがあって、それぞれ2アイテムづつ、2行の数字があるのは、１行目はラインをカード上に描き始めたポイントのXY座標、2行目は描き終えたポイントのXY座標です。



ラインのポイントは「points　ポイント（複数）」を使って求められます。返される値は、描き始めと終わりの2カ所の座標が、2行で返されます（下図）。

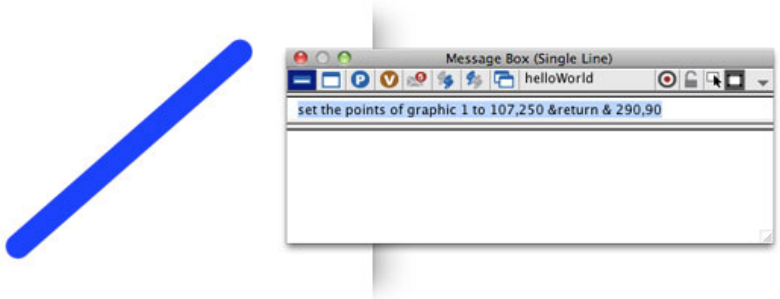
```
put the points of graphic 1
```
グラフィック１のポイント（複数）を返しなさい



これは「始めのXY座標 & 改行 & 終わりのXY座標」と言う内容ですから、LiveCodeの言葉で書き直すと「107,97 & return & 291,283」となります。ここで書いた「return　リターン」は改行です。LiveCodeで使える改行は他にも幾つかの言い方がありますが、ここではリターン・キーと言う意味の「return」が分かりやすい英語なので、リターンを使うことにして、反対の傾斜にする設定をスクリプトで送ります。

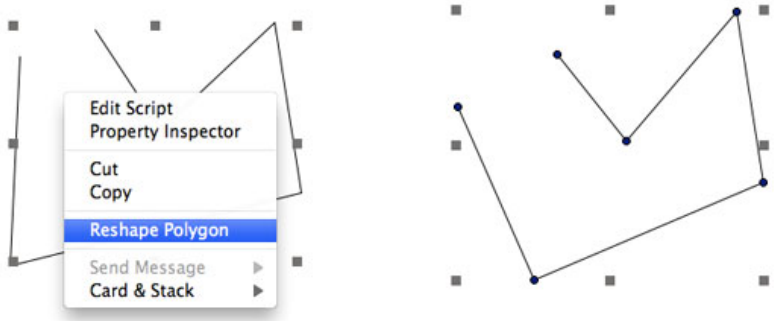
```
set the points of graphic 1 to 107,250 & return & 290,90
```

グラフィック１のポイント（複数）を「107,250 &　改行　& 290,90」に設定しなさい



これで「points　ポイント（複数）」と「return　改行」の使い方が理解できましたか。「&記号」は「04章: 測定と位置の設定」の使い方と同じです。

カード4は「Freehand Polygon　フリーハンド・ポリゴン」を描きます。ツール・パレットのベクター・グラフィック作成の2列目一番右です。クリックをしてドラッグ、クリックをしてドラッグを6〜7回続けます。最後はダブル・クリックをすると、マウスについてきたラインを切り離すことができます。描き終えたらインスペクタを見ると、カード3でラインを描いた時のようにポイントの座標が見えます。この座標でフリーハンド・ポリゴンのポイントを変えて行く事もできますが、視覚的にマウスで動かす方法もあります。下図左のように、オブジェクトのコンテクストメニューから「Reshape Polygon　リシェープ・ポリゴン（多角形の変形）」を選ぶと、下図右のようにポリゴンの各コーナーに小さな丸が現れて。マウスで移動できるようになります。



フリーハンド・ポリゴンに、自分の好きな色をいろいろ試してください。ボーダーもあってもなくても良いです。 ブラウザー・ツールに持ち替えて、カードの左右に置いたボタンのクリックで、カードを次々に開いてみましょう。

スタックを保存しておきます。「カード1」を開いてから、「Fileメニュー > Save」を選びます。スタックを終了させて、次に開く時は「カード1」が開かれますから、右上のテキスト・フィールドの数字を「１」にしておかないと、ページ・ナンバーが違ってしまいます。次は

カードの移動で「ビジュアル・イフェクト　Visual Effect」をかけます。

練習問題

Fileメニューから新しくメイン・スタックを作って、「Push Button」をツール・パレットからカードの下の辺りに置き、ボタンのマウスアップ・メッセージ・ハンドラーの中に、この06章で作ったカード１と同じレクタングルを作るスクリプトを書いて、マウス・クリックでオブジェクトを作りなさい。グラフィックの輪郭のカラーは「forestGreen」、中のカラーは「yellow」とします。
ヒント：始めの行は「create graphic」を使います。作ったグラフィックは「graphic 1」です。作るグラフィックの「style」は「rectangle」です。　解答は次の章にあります。

Tips

- プロパティの真偽（true, false）の変更をした場合、インスペクタは一旦閉じるか、ページを変えてもう一度開き直すかしないと、正しい表示がされません。
- ふたつの言葉を合わせて作った言葉は、二つ目の言葉の始めを大文字にしておくと理解しやすくなります。たとえば「lineSize」「foregroundColor」等。
- 色彩の微妙な設定には、カラー・ネームよりRGBカラーが適しています。
- オブジェクトが透明でセレクトできない時は、オブジェクトの外側からマウスを対角線にドラッグします。
- グラフィックの「レクタングル」とか「ライン」とかの型の違いは、「style　スタイル」と呼びます。
- ポイントの各行をつなぐ言葉は「return　リターン」を使います。
- スタックを保存する場合は、次にユーザーが開いた時に全てが始めの状態になるようにしておきます。

この章で出て来た言葉

opaque
プロパティ（property）コントロールの内側が不透明か透明かを特定する
true
コンスタント（constant）プロパティの値・表現が正しい
false
コンスタント（constant）プロパティの値・表現が違っている
`set the opaque of graphic 1 to true`

```
set the opaque of graphic 1 to false
```

lineSize
プロパティ（property）オブジェクトの輪郭のサイズを特定する
```
set the lineSize of graphic 1 to 10
```

lockLocation
プロパティ（property）ユーザーがコントロールを動かす事ができるか　短縮形：lockLoc
```
set the lockLocation of graphic 1 to true
set the lockLoc of graphic 1 to falsee
```

foregroundColor
プロパティ（property）オブジェクトの輪郭線の色彩を特定する
```
set the foregroundColor of graphic 1 to 255,255,0
set the foregroundColor of graphic 1 to Blue
```

backgroundColor
プロパティ（property）オブジェクトの内側の色彩を特定する
```
set the backgroundColor of graphic 1 to 90,255,0
set the backgroundColor of graphic 1 to Yellow
```

colorNames
ファンクション（function）LiveCodeで使えるカラー・ネームを返します
```
the colorNames
```

style
プロパティ（property）　オブジェクトを特定する型
roundRect
キーワード（keyword）グラフィックやボタンの角の丸い形　同義語：roundRectangle
```
set the style of graphic 1 to roundRect
```

line
キーワード（keyword）線状のグラフィックの型（ポイントを設定しないと透明になります）
```
put the style of graphic 1
set the points of graphic 1 to 50,40 & return & 250,300
```

points
プロパティ（property）グラフィックの各頂点の座標
return
コンスタント（constant）改行を意味する言葉
```
set the points of graphic 1 to 107,250 & return & 290,90
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

07: ビジュアル・イフェクトと条件文

この章の概略

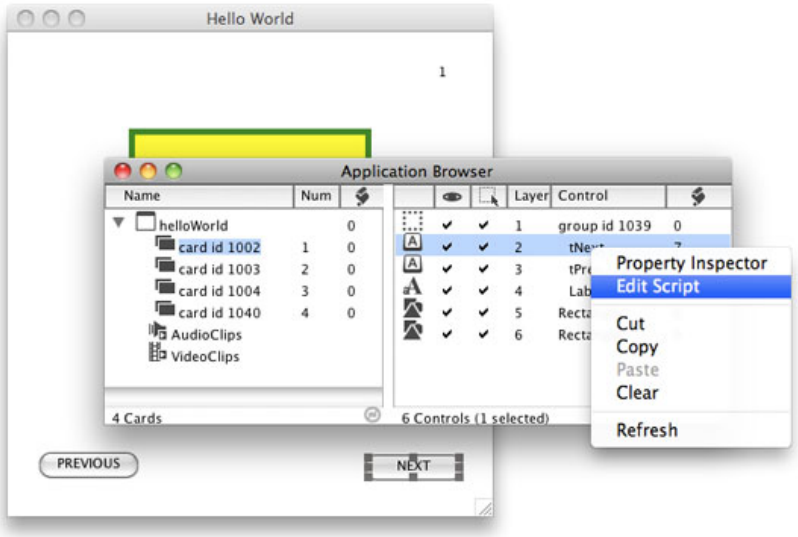
- *カード間の移動で使うビジュアル・イフェクトと、カード上のオブジェクトの位置や形が変化した時に使うビジュアル・イフェクト。*
- *ボタン・メニューのレイベル名を「get」を使ってスクリプトに取り込む。ある値を一時的に収納する文字列（variable　バリアブル）として使えない名前。*
- *「if」を使った条件文で、オブジェクトを見せる、隠す。*
- *最後に06の練習問題の解答があります。*

カード間の視覚効果

スタック「helloWorld.livecode」のカード移動に、ビジュアル・イフェクトをつけます。例えば右から左にカード上のグラフィックが移動したように見せたり、次のカードの画像がボンヤリと現れたりとか、様々な視覚的な効果ができます。始めは「wipe left」と「wipe right」を使ってみましょう。「ボタン・レイベル NEXT」のスクリプト・エディターを開きます。エディット・ツールで「ボタン・レイベル NEXT」をセレクトしようとすると、バックグラウンドにしているグループ全体がセレクトされてしまいます。グループのスクリプト・エディターなら直接ダブル・クリックで開けられますが、グループの中のコントロールはセレクトできません。毎回グループの編集機能を使わなくてはいけないとなると、ちょっと面倒ですね。でも、これには二つ方法があります。
１）まずブラウズ・ツールで
Mac OSなら「コマンド・キー + オプション・キー」
Windowsなら「コントロール・キー + オルト・キー」を押しながら、
目的のコントロールをクリックすると、スクリプト・エディターが開きます。
２）もう一つは、アプリケーション・ブラウザーから、目的のコントロールをセレクトしてコンテクスト・メニューで「Edit Script」を選びます。

この章で使われる英単語

- visual [形容詞：ビジュアル, 視覚の]
- effect [名詞：効果, 影響]
- wipe [動詞：拭く, 消す]
- checkerboard [名詞：市松模様]
- venetian blinds [名詞：ベネチア風な日よけのウライインド]
- iris [名詞：カメラの絞り]
- dissolve [動詞：(物が）溶ける, （画面が）少しづつ重なって入れ替わる]
- option [名詞：選択肢]
- get [動詞：ゲットする, 得る]
- expression [名詞：表現, 語句,（数学の）式]
- variable [名詞：変化するもの, ある値を収納する文字列]
- Oval [名詞：楕円形, 長円形]
- side(s) [名詞：側面, （幾何図形の）辺]
- visible [形容詞：目に見える, 可視の]
- invisible [形容詞：目に見えない, 不可視の]
- show [動詞：見せる, 見える]
- hide [動詞：隠す]
- if [接続詞：もし ... ならば]
- then [副詞：それなので, 従って]
- else [副詞：さもなければ]



「ボタン・レイベル NEXT」のエディターが開いたら、すでに書き込んであるマウスアップ・ハンドラーのスクリプトの一番上に「visual effect "wipe left"」入れて、「Apply」をクリックします。マウスアップのメッセージ・ハンドラーから全部下に書きました。（日本語部分を書き込むとエラーになります）

```
on mouseUp
   visual effect "wipe left"
   ビジュアル・イフェクト「wipe left（ワイプ　レフト）」
   （この行を始めに入れる。以下は同じ）

   go next card
   put the number of this card into field 1
end mouseUp
```

これでボタン「NEXT」をブラウズ・ツールでクリックすると、グラフィックを左に向かって消しながら、新しいグラフィックが出て来る視覚効果「wipe left（ワイプ　レフト）」になります。

同じようにグループの中にある「ボタン・レイベル PREVIOUS」のエディターを開いて、すでに書き込んであるマウスアップ・ハンドラーのスクリプトの一番上に、今度は反対方向にイフェクトが見える「visual effect "wipe right"」入れて、「Apply」をクリックします。こちらも、マウスアップのメッセージ・ハンドラーから全部下に書きました。（日本語部分を書き込むとエラーになります）

```
on mouseUp
   visual effect "wipe right"
   ビジュアル・イフェクト「wipe right（ワイプ　ライト）」
   （この行を始めに入れる。以下は同じ）

   go prev card
   put the number of this card into field 1
end mouseUp
```

これで「PREVIOUS」をクリックするとグラフィックを右に向かって消しながら、新しいグラ

フィックが出て来る視覚効果「wipe right（ワイプ　ライト）」になります。

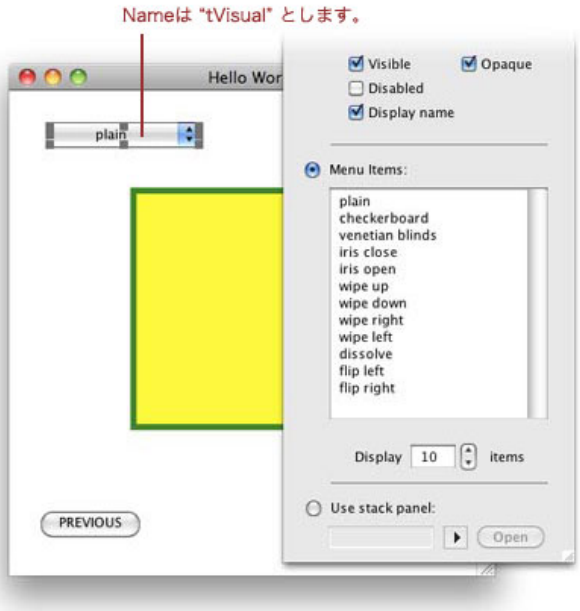
イフェクトのサンプルが選べるボタン・メニュー

LiveCodeでは、沢山ののビジュアル・イフェクトが用意されています。バックグラウンドにオプション・ボタンのメニューを置いて、ユーザーがイフェクトを選択できるようにします。ビジュアル・イフェクトの種類を実際に見る事ができて、ボタン・レイベルに表示されたイフェクト名を、スクリプトに取り込む方法の学習です。ここでは10のイフェクトの種類を下にリスト・アップします。これ以外のイフェクトはLiveCodeの辞書を見てください（辞書については後の章で説明します）。「plain　プレイン」は、何のイフェクトも施さない言葉です。

```
plain
checkerboard
venetian blinds
iris close
iris open
wipe up
wipe down
wipe right
wipe left
dissolve
```

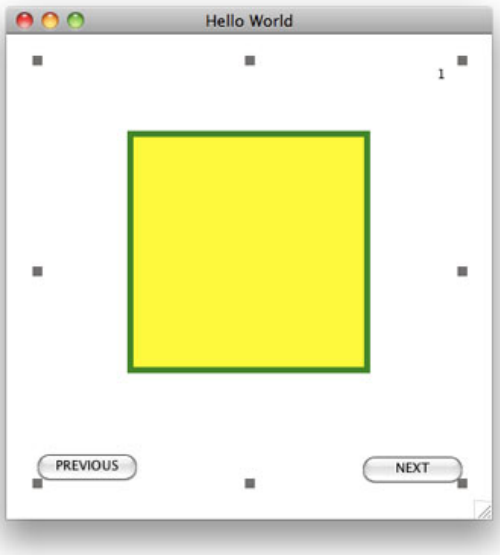
カードの左上に、ツール・パレットの各種メニュー・ボタンの作成から「Option Menu　オプション・メニュー」を置いてください。後からバックグラウンドにしているグループに組み込みますから、今は作業がしやすいように、どのカード上に置いても良いです。メニュー・ボタンは、普通のアプリケーション上部にあるメニュー・バーとは少し違っていて、カード上のどこにでも置く事ができます。オプション・メニューは、ユーザーが選んだメニューの項目がボタンのレイベル（label）に現れるので、レイベル名をスクリプトで「get　ゲット」して、そのイフェクト名をスクリプト中に組み込みます。

新しく作ったオプション・メニューを、エディター・ツールを使ってダブル・クリックでインスペクタを開いたら、名前（name）を「tVisual」とします。インスペクタの下の方に、「Menu Items」と言うテキストを打ち込めるフィールドがありますから、上にリスト・アップしたイフェクト名をペイストします（下図）。ブラウズ・ツールに持ち替えて、オプション・メニュー「tVisual」をクリックすると、セレクトできるメニューが現れます。



Windowsでは、インスペクタの「Display」を「10 items」にしてください。
デフォルトの「5」は、メニューが5行だけ表示されます。

もう一度エディット・ツールに持ち替えて、ボタン名「tVisual」をセレクトしてカットしたら、バックグラウンドのグループを選んで、05章で扱ったグループの編集機能を使って（下図）バックグラウンド・グループの中にボタン「tVisual」をペイストします。グループ編集中は、グループ内にあるコントロールだけが見えるようになりますから、レイヤーのトップにあるグラフィックは見えない状態になります。グループ編集を解除して、もう一度ブラウズ・ツールに持ち替えると、全てのカードでこのボタン・メニューが使えるようになります。



ブラウズ・ツールとエディット・ツール（ポインター・ツール）を
頻繁にツール・パレットで変更するのが面倒なら、ショート・カットが使えます。
ブラウズ・ツールは、MacOS: コマンド+9, Windows: コントロール+9

エディット・ツールは、MacOS: コマンド+0, Windows: コントロール+0

「get」の意味と使い方

「get　ゲット」でボタンのレイベルを読み取って、スクリプトの中のビジュアル・イフェクトを入れ替えます。「get」が読み取ったボタン・レイベル（value = 値）は「it（itと言う名前の器 = variable　バリアブル）」の中に収納されます。

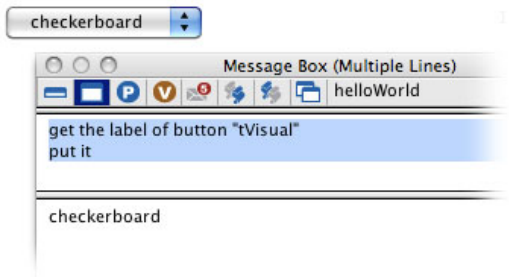
```
put （expression ある内容の値・数字・文字列） into it
```

「get」はこのスクリプトと同じ意味の、簡略した言い回しです。

「NEXT」ボタンの中のスクリプトに書き込む前に、マルチライン・メッセージ・ボックスで試してみます。マルチライン・メッセージ・ボックスは複数行をタイプしたら「エンター・キー」を叩きます。リターン・キーとエンター・キーがひとつになっているキーボードは、「コントロール・キー + リターン・キー」を叩きます。

```
get the lablel of button "tVisual"
put it
```

ボタン「tVisual」のレイベルをゲットしなさい（レイベル名を「it」の中に入れなさい）。
「it」をメッセージ・ボックスに書き出しなさい。



メニュー・ボタンのレイベルを変えて、「it」に入れた値をメッセージ・ボックスに書き出してください。「get」で得たレイベル名は、いつも「it」に入って書き出されます。

それでは「NEXT」ボタンの中に「it」を使ったスクリプトを書き込みます。ブラウズ・ツールでMac OSは「コマンド・キー + オプション・キー」、Windowsは「コントロール・キー + オルト・キー」を押しながら「NEXT」ボタンをクリックして、スクリプト・エディターを開いて、下のスクリプトを書き込みます。（日本語を書き込むとエラーになります）

```
on mouseUp
   get the label of button "tVisual"
   visual effect it
   ボタン「tVisual」のレイベル名をゲット（itに入れる）しなさい
   ビジュアル・イフェクト　「it （= レイベル名)」
   （この赤文字2行を始めに入れる）

   go next card
   put the number of this card into field 1
end mouseUp
```

ボタン名「PREVIOUS」にも同じスクリプトを書き込みます。

```
on mouseUp
   get the label of button "tVisual"
   visual effect it
   ボタン「tVisual」のレイベル名をゲット（itに入れる）しなさい
   ビジュアル・イフェクト 「it (= レイベル名)」
   （この赤文字2行を始めに入れる）

   go prev card
   put the number of this card into field 1
end mouseUp
```

それでは各種イフェクトを選んで、カードを開いて効果を試してください。

ここでは「get」を使ってボタン・レイベルを読み取り「it」に収納しましたが、「put ... into ある文字列（variable　バリアブル）」と言う方法でもかまいません。例えば、

```
put the label of button "tVisual" into tButtonLabel
visual effect tButtonLabel
  ボタン「tVisual」のレイベルを「tButtonLable（ある文字列の器 = variable）」に入れなさい。
  ビジュアル・イフェクト tButtonLabel（ある文字列の器 = variable　バリアブル）
```

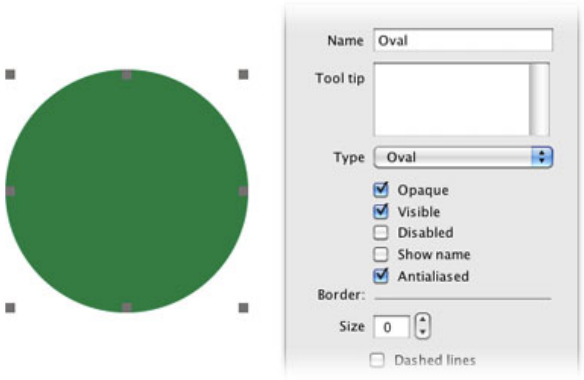
この例では、ある文字列（variable　バリアブル)を「tButtonLable」としています。バリアブル（variable）は、毎回違う数字や文字列などの値を、一時的に収納する器（文字列）と考えれば良いでしょう。数字で始まる文字列にはできません。「&」が入っている文字列は使えません。スペースのある1語以上の文字列は使えません（必ずシングルワードにします）。日本語の文字は使えません。

普通のプログラミングの用語では「variable」を「変数」と言う日本語にする事が多いようですが
LiveCodeでの訳語としては、しっくり来ないので「バリアブル」としています。

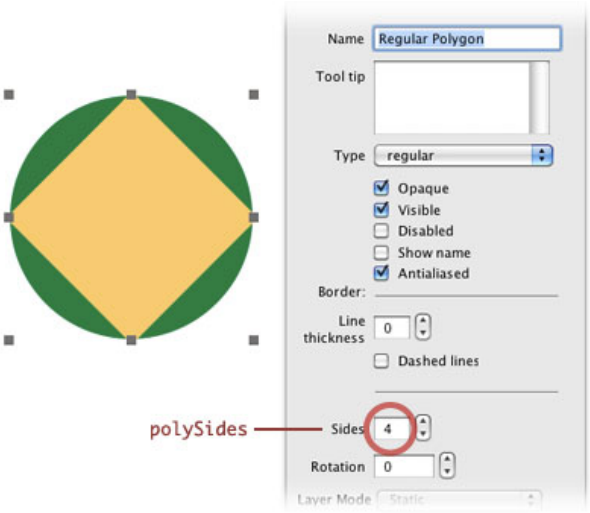
円「Oval」と普通の多角形「Regular Polygon」の作成

今度はカードを移動させないで、オブジェクトを消したり見せたりする時につける、ビジュアル・イフェクトをやってみます。始めにその為の新しいカードとグラフィック2点を作ります。カード4を開いたら、「Objectメニュー > New Card」で、カード4の後に新しい「card 5」を作ります。カード5には二つのグラフィックを作って、交互にグラフィックが見えるようにして、それにビジュアル・イフェクトをかけます。

ツール・パレットの「ベクター・グラフィック」作成から、上の列右から二つ目「Oval　オーバル」を選んで、カード5の上に横「width = 200」、縦 「height = 200」、ロケーション「200,180」、「Opaqu　オペーク」、ボーダー・サイズ「lineSize = 0」、バックグラウンド・カラーは任意の緑の、円形（graphic 1　グラフィック1）を作ります（下図）。オーバルは意味では長円形ですが、グラフィックは正円にしました。



もう一つ緑の円の上に、こんどはツール・パレットの「ベクター・グラフィック」作成から、上の列一番右の「Polygon　ポリゴン」を選んで、横「width = 210」、縦 「height = 210」、ロケーション「200,180」、「Opaqu　オペーク」、ボーダー・サイズ「lineSize = 0」、バックグラウンド・カラーは任意の黄色の、多角形（graphic 2　グラフィック2）を作ります。デフォルトの「Regular Polygon　レギュラー・ポリゴン」は12角形ですから、各数を減らして行きます。インスペクタの下の方に「Sides（辺）」があります。「Sides（辺）」を「4」にします。普通の英語で「sides」は幾何学図形の辺ですが、正確なLiveCodeのプロパティの言葉では、多いと言う意味の「poly」を付けて「polySides」と言います。フリーハンド・ポリゴンではポイント（points）の座標を設定しましたが、普通の多角形では辺（polySides）の数で設定します。



条件文を使ってオブジェクトを見せる、隠す

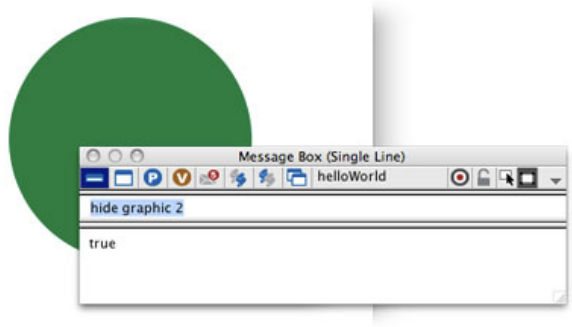
「graphic 2　グラフィック2」を見えなくします。インスペクタの設定は「visible ビジブル（可視）」を「true」か「false」にするかの設定になっていますから、プロパティの「visible」を「false」にすれば、オブジェクトは見えなくなります。別な表現で言えばプロパティの「invisible　インビジブル（不可視）」を「true」にすれば、やはりオブジェクトは見えなくなります。同じようにオブジェクトを「show　ショー（見せる）」「hide ハイド（隠す）」と言うコマンドで、見せたり、隠したりもできます。下の3行はどれも同じ指示を与えて、グラフィック2を見えなくします。

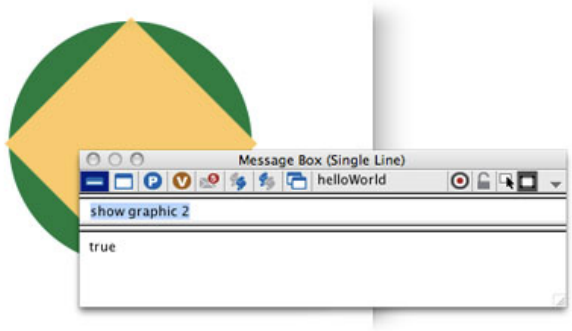
```
set the visible of graphic 2 to false
set the invisible of graphic 2 to true
hide graphic 2
```
（どれも）グラフィック２を隠しなさい（の意味です）

反対にグラフィック2を見えるようにするには、

```
set the visible of graphic 2 to true
set the invisible of graphic 2 to false
show graphic 2
```
（どれも）グラフィック２を見せなさい（の意味です）

私の場合「visible ビジブル（可視）」「invisible　インビジブル（不可視）」は、オブジェクトが見えているか見えていないかの条件の文で、オブジェクトの可視・不可視には「show ショー（見せる）」「hide ハイド（隠す）」を使っています。





さてこの「visible, show, hide」を「if もし...ならば」と言う条件の文に当てはめて使ってみましょう。始めに「if」の基本的な文の構造を書いておきます。LiveCodeではこの文の構造を「syntax　シンタックス」と呼んでいます。

簡単な1行で済むような指示なら「if（もし）」の後に「then（だとしたら）」を付けて

```
if ある条件の文章 then（条件が真なら）この指示を実行する
```

そうでなく、他の場合にはこうすると言う、もう一つの指示を与える場合には「if ... then」の次に「else（さもなければ）」の指示を書いて、最後は必ず「end if」で閉めます。

```
if ある条件の文章 then（条件が真なら）
    この指示を実行する
```

```
  else（さもなければ＝条件が偽なら）
    別な指示を実行する
end if
```

では上で作った二つのグラフィックでやってみましょう。新しく「push button　プッシュ・ボタン」をカードの中央下に作ります。プロパティはデフォルトのまま、スクリプト・エディターを開きます。マウスアップ・ハンドラーがありますから、その中に下のスクリプトを書き込みます。

```
if the visible of graphic 2 then
    hide graphic 2
  else
    show graphic 2
end if
```

このグラフィックの「show」「hide」にビジュアル・イフェクトを付けます。カード間の移動では「go next card」の上に「visual effect "イフェクト名"」を入れました。今回はカード間の移動がないので、「if ... then ... else ... end if」が始まる前に、一連の動作が実行される様子の画面を、ビジュアアル・イフェクトの為に「lock scree　ロック・スクリーン」で一旦止めて、実行が終わった時点で「unlock scree　アンロック・スクリーン」と同時にビジュアル・イフェクトをかけます。

```
on mouseUp
  lock screen for visual effect
  if the visible of graphic 2 then
      hide graphic 2
    else
      show graphic 2
  end if
  unlock screen with visual effect "dissolve"
end mouseUp
```

これでカードの移動なしにでも、ビジュアル・イフェクトをかけながら、画面に効果をつける事ができます。

06章の練習問題答え

問題： Fileメニューから新しくメイン・スタックを作って、「Push Button」をツール・パレットからカードの下の辺りに置き、ボタンのマウスアップ・メッセージ・ハンドラーの中に、この06章で作ったカード１と同じレクタングルを作るスクリプトを書いて、マウス・クリックでオブジェクトを作りなさい。グラフィックの輪郭のカラーは「forestGreen」、中のカラーは「yellow」とします。

```
on mouseUp
   create graphic
   set the style of graphic 1 to rectangle
   set the width of graphic 1 to 200
   set the height of graphic 1 to 200
   set the location of graphic 1 to 200,180
   set the lockLocation of graphic 1 to true
```

```
   set the opaque of graphic 1 to true
   set the lineSize of graphic 1 to 5
   set the foregroundColor of graphic 1 to forestGreen
   set the backgroundColor of graphic 1 to Yellow
end mouseUp
```

Tips

- グループ内のコントロールのスクリプト・エディターを開くには、ブラウズ・ツールで Mac OSなら「コマンド・キー + オプション・キー」Windowsなら「コントロール・キー + オルト・キー」を押しながら、 目的のコントロールをクリックします。

- ボタン・レイベルを読み取るのは「get」を使って「it」に収納しましたが、他の「バリアブル　variable（文字列の器）」の中に「put ... into 」を使って入れる事もできます。「バリアブル　variable（文字列の器）」は、数字で始まる語にはできません。&が入っている語は使えません。1語以上の名前は使えません（必ずシングルワードにします）。日本語の文字は使えません。

- ツールを持ち替えるショート・カットは、ブラウズ・ツールの場合、MacOSでは コマンド＋9　Windowsでは コントロール＋9 。エディット・ツールの場合、MacOSでは コマンド＋0　Windowsでは コントロール＋0 。

この章で新しく出て来た言葉

visual effect
コマンド（command） オブジェクトの変化に視覚効果を加える
```
visual effect "wipe left"
visual effect "wipe right"
```

get
プコマンド（command)ある表現（式）を「it」に入れる
```
get the lablel of button "tVisual"
```

polySides
プロパティ（property)レギュラー・ポリゴンに幾つの辺があるか
```
put the polySides of graphic "regular polygon" into numerSides
```

visible
プロパティ（property)オブジェクトが見えていれば「真」、隠れていれば「偽」を特定する
invisible
プロパティ（property)オブジェクトが隠れていれば「真」、見えていれば「偽」を特定する
```
set the visible of graphic 2 to false
set the invisible of graphic 2 to true
```

lock screen
コマンド（command)プロパティ「lockscreen」を一時的に真にする
```
lock screen for visual effect
```

unlock screen
コマンド（command）プロパティ「lockscreen」を一時的に偽にする

```
unlock screen with visual effect "dissolve"
```

show
コマンド（command）オブジェクトを見えるようにする

```
show graphic 2
```

hide
コマンド（command）オブジェクトを見えなくします

```
hide graphic 1
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

08: 算数クイズ 1

この章の概略

- *LiveCodeの足し算、引き算、掛け算、割り算の小学校の算数との表記の違い。*
- *「足し算クイズ」のインターフェイス、問題作り、解答合わせのスクリプト。*
- *簡単なダイアログ・ボックス。*

30年程前までは、コンピュータの事を電子計算機と言っていたくらいですから、小学生程度のやさしい算数クイズを作ってみましょう。ただの計算機では面白みがないので、プログラミングで問題を作って、ユーザーの答えに対話（ダイアログ）で回答します。まだ前章までのスタック「helloWorld」を開いていたら、「Fileメニュー > Close and Remove From Memory」を選びます。スタックのウインドウを閉じただけでは、LiveCodeの中に、スタックのメモリーが残ったままになってしまいますから、メモリーも同時に削除します。

この章で使われる英単語

- warp [動詞：包み込む, テキストを自動改行する]
- random [形容詞：成り行きの, 無作為の]
- exit [動詞：退場する, から抜ける]
- send [動詞：送る]
- asterisk [名詞：アスタリスク, *星印]
- divisor [名詞：割り算の割る方の数, 除数]

足し算、引き算の表記

足し算はプラス記号（+）引き算はマイナス記号（-）を使うのは、普通の算数と同じです。簡単な計算をやってみましょう。1行づつメッセージ・ボックスで試してください。

```
2 + 3
100 + 200
65 - 30
300 - (380 -100)
```

普通の算数と同じように答えの数字が返されます。カッコがあるとその中が始めに計算されるのも同じです。では次に答えもある式を書いてみます。

```
2 + 3 = 5
100 + 200 = 330
65 - 30 = 35
300 - (380 -100) = -180
```

イコール記号の使い方も、普通の算数の表記と同じです。上から「true」「false」「true」「false」と返されます。書いてある行が、式として成り立っているかを「真偽」で答えますと言っても良いし、答えが正しいか間違っているか、と言う言い方でももちろん良いです。イコール記号（=）は、普通の英語の文の「is」にも書き換える事ができます。「is not」と言う表現もできますが、話がややこしくなるので、ここでサンプルを書くのはやめておきます。

```
2 + 3 is 5
100 + 200 is 330
```

```
65 - 30 is 35
300 - (380 -100) is -180
```

読みやすいように始めの計算部分をカッコで囲っても同じ回答が得られます。

```
(2 + 3) is 5
(100 + 200) = 330
(65 - 30) = 35
(300 - (380 -100)) is -180
```

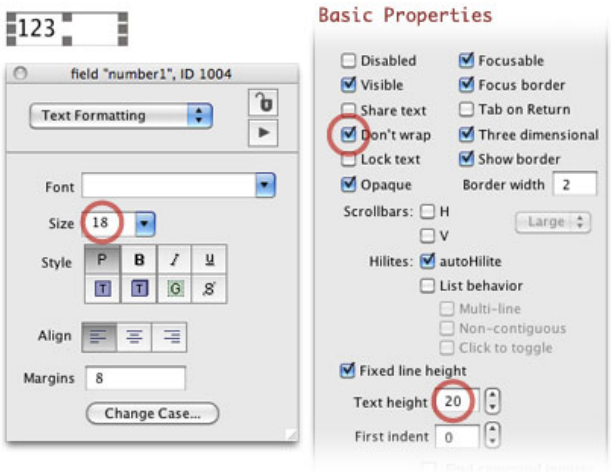
少数も同じに計算できますが、少数点第7位の結果は四捨五入されます。別な言葉で言えば、少数点第7位までの計算をします（ただし答えは第6位までです）。

```
3.14159265 + 20
```

少数点第7位が四捨五入されて `23.141593` が返されます。

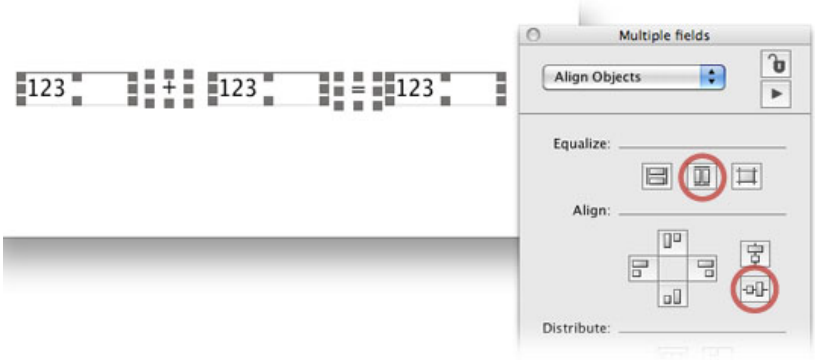
「足し算クイズ」のインターフェイス作成

新しいメイン・スタックをメニューから作ります（Fileメニュー＞New Mainstack）。スタックの名前は「sansuu」にしましょうか。ツール・パレットのテキスト・フィールドから「Text Entry Field」をひとつ、カード上にドラッグ＆ドロップします。文章をタイプする普通のテキスト・フィールドです。マウスをツール・パレットに持って行くと、「Text Entry Field」と書かれた、黄色のツール・チップが出てきます。フィールドのデフォルトのフォント・サイズは11ポイントですから、大きなフォントが入るように、フィールドの天地を少し伸ばします。
フィールドのプロパティ・インスペクタを開けて、ベーシック・プロパティ（Basic Properties）で名前は「number1」とします。「Don't Wrap（改行しない）」をチェック（true）して、18ポイントのテキストを入れる予定ですから、天地がちょうど良いアキになるよう「Text height（行間）」を「20」ポイントにしてください。プロパティ・インスペクタの「Text Formatting」の「Size」はデフォルトのフォント・サイズ（11ポイント）で空白になっていますから「18」ポイントにします。「Font」がデフォルトで空白になっているのは、OSのシステム・フォントです。デフォルトのフォント名は、Mac OSでは「Lucinda Grande」、Windowsでは「Segoe UI」です。

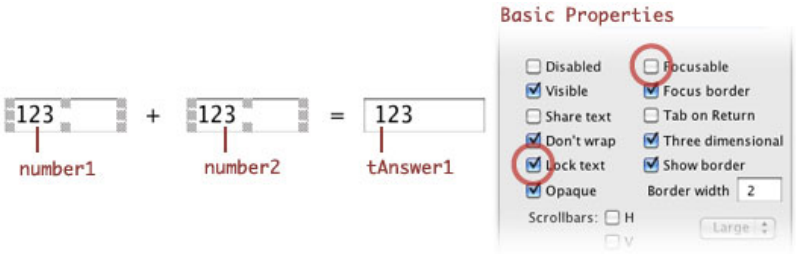


同じプロパティのテキスト・フィールドをあと２つ、オプションキーを押しながらドラッグしてコピーを作ります。名前は「number2」と「tAnswer1」にします。ツール・パレットからテキスト・フィールドの「Label」をドラッグ＆ドロップして、フィールド「number1」と同じフォント・サイズ、テキスト・ハイトにして、「Text Formatting」の「Align」はセンターにします。このフィールドは直接ブラウズ・ツールでタイプできないので、プロパティ・インスペクタの「Contents」を開いて「プラス記号」をタイプします（下図）。「プラス記号」のフィールドをもうひとつ、Mac OSなら オプション・キー + ドラッグ＆ドロップ（Windowsはオルト・キー + ドラッグ＆ドロップ）でコピーを作って、今度は「イコール記号」にします。

5つのテキスト・フィールドを下図のように適当な間隔で配置したら（スタックの横巾も適当に広げてください）、全部をポインター・ツールでセレクトして、インスペクターの「Align Objects」で高さを揃えて、きちんと横に並んでいるようにアラインします（下図）。



フィールド「number1」と「number2」をセレクトして、「Basic Properties」の「Lock text」を「true（チェックする）」、「Focusable」を「false（チェックを外す）」にセットします（下図）。この設定で「number」の2つのフィールドにはユーザーはタイプできなくなって、「tAnswer1」にだけタイプできるようになります。



ランダム「random」に数字を選ぶ

日本語でもランダムと言う言葉は使うので、だいたい予測はつくと思います。「the random of 上限の数字」で数字が「10」でしたら、「1から10までの数」から無作為に数字が返されます。例によってメッセージ・ボックスでやってみましょう。

```
put the random of 10
```

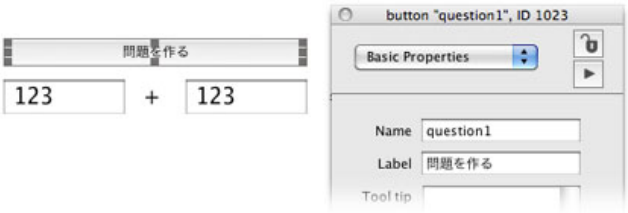
1から10までの数字を無作為に選んで返しなさい。

上限がそう大きくないので、何度か同じ数字が返される事もあります。今はできるだけ日常英語のような表現を使っていますが、だいぶLiveCodeに慣れて来ると、この後の章で出て来るファンクションのような表記にしたくなる人達もいるので、上限の数字をカッコに入れるだけでも同じ意味にできるようになっています。

```
put random(10)
```

1から10までの数字を無作為に選んで返しなさい。
（これも上で書いた「random」の表現とまったく同じ意味です）

ランダムに数字を発生させて、それをテキスト・フィールドに入れて問題とします。その為のボタンを作ります。ツール・パレットのボタンから四角い「Rectangle Button」をカード上に置いて、名前（name）は「question1」、レイベル（label）は日本語で「問題を作る」にします（下図）。ボタンの「Name」は、スクリプト中で使うので日本語にできませんが、これから使うこのボタンの「Label」は表示だけですから、インスペクタで日本語のレイベルに設定しても問題ありません。



ボタン名「question1」のスクリプト・エディターを開きます。ボタン名「question1」をエディット・ツールでセレクトしたら、アイコン・メニューの「Code」をクリックするか、コンテキスト・メニューから「Edit Script」。またはそれ以外の方法でも良いです。ボタンのエディターには「マウスアップ・ハンドラー」が既に用意されています。

```
on mouseUp
```

```
   put the random of 100 into field "number1"
   put the random of 100 into field "number2"
   put empty into field "tAnswer1"
end mouseUp
```

スクリプト・エディターは書いてある内容が読みやすいように、
ハンドラーの中の行の頭に空白（インデント）を取るようになっていますが、
もしインデントがうまく表示されていないようでしたら、
行のどこかにカーソルを入れて「タブ・キー」を叩きます。

始めの2行は上に説明があるように、1から100までの数字をランダムに作り出して、ふたつのテキスト・フィールドに入れます。3行目にある「empty」は「空っぽ」と言う意味の英語ですから、テキスト・フィールド「field "tAnswer1"」にあるすべてを消して、空っぽをフィールドに入れます（つまり消し去ります）。上では「empty」と書いていますが、ダブル・クオートだけの中身の何もないストリング「""」でも同じです。

```
put "" into field "tAnswer1"
```

答えが正しいかどうか調べるボタンを作る前に、正しい解答を得る方法をマルチライン・メッセージ・ボックスで見てみます。

```
put field "number1" into tNumber1
put field "number2" into tNumber2
put field "tAnswer1" into tAnswer1
put tNumber1 & return & tNumber2 & return & tAnswer1
```

上記をマルチライン・メッセージに書いたら、エンター・キーを叩きます（またはコントロール・キー + リターン・キー）。フィールドにある各数字を、前章で説明したバリアブル（tNumber1, tNumber2, tAnswer1）に入れて、その3つのバリアブル（variable)をメッセージ・ボックスに3行で書き出しています。

バリアブルは、毎回違う数字や文字列などの値を、一時的に収納する器（文字列）です。
数字で始まる文字列にはできません。「&」が入っている文字列は使えません。
スペースを挟んだ1語以上の文字列は使えません。日本語の文字は使えません。

```
put tNumber1 + tNumber2 = tAnswer1
```

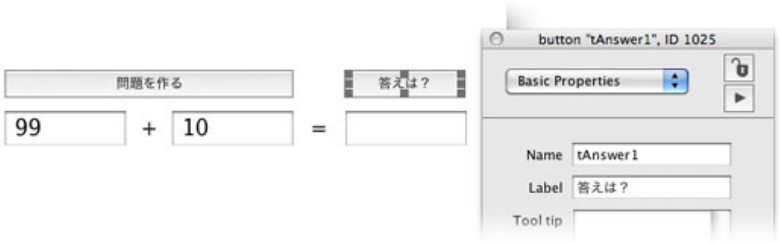
始めに普通の算数式で説明したように、上のこのスクリプトが「true　真」ならば、答えは正しく、「false　偽」ならば間違っています。マルチライン・メッセージに書き込んだ最後の行を、これに入れ替えてみます。

```
put field "number1" into tNumber1
put field "number2" into tNumber2
put field "tAnswer1" into tAnswer1
put tNumber1 + tNumber2 = tAnswer1
```

これで「true」が返されれば答えは正しく、「false」が返されたら間違った答えと言う事になります。この正しいか違っているかを、これも前章で学んだ条件の「if ... then」で判断すれば良い訳です。その部分を書いてみましょう。

```
if tNumber1 + tNumber2 = tAnswer1 is true then
    put "Very Good."
else
    put "Sorry."
end if
```

メッセージ・ボックスからはこれくらいにして、ボタンを作ってその中にスクリプトを書いて行きます。ボタン「question1」と同じ「Rectangel Button」を作ります。名前（Name）は「tAnswer1」、レイベル（Label）は「答えは？」にします（下図）。



上のメッセージ・ボックスで試した「if条件文」では、丁寧な言い方をして「tNumber1 + tNumber2 = tAnswer1 is true」と書いていますが、「true」と言う前提の文では「is true」を省いて書く事もできます。それでは「answer」を使ってユーザーと対話できるようにします。「answer」はダイアログ・ボックスでユーザーと対話形式のやり取りができるコマンドです。ここでは「OK」だけの回答にしますが、幾つかの選択を用意することもできます。日本語のダイアログも書けますが、もう少し学習のステップを踏んでからの方が良いので、今は英語にします。それから間違っていた場合にビープ音を鳴らすように、「beep」も入れる事にしましょう。以下を今作ったボタン「tAnswer1」の中に書き込みます。

```
on mouseUp
   put field "number1" into tNumber1
   put field "number2" into tNumber2
   put field "tAnswer1" into tAnswer1
   if tNumber1 + tNumber2 = tAnswer1 then
      answer "Very Good!"
   else
      beep
      answer "Sorry."
   end if
end mouseUp
```

数字かどうかを判断、リターン・キーを無効にする

これで「足し算クイズ」としての機能は完成ですが、アプリケーションとしてはユーザーがちょっとした間違いなどをするといけないので、その対策が必要です。ひとつは解答欄に数字以外の文字を打ち込むかもしれません。もうひとつは解答欄でリターン・キーを叩くと複数行の答えになってしまいます。

解答欄にある文字列が数字かどうかと言う判断は「isNumber」を使います。これは正しい意味

で英語ではありませんが、ニュアンスとして「これって数字？」と言う感じでしょうか。「isNumber」は、LiveCodeに類似する他の日常英語のようなプログラミング言語、SuperTalkでも使われています。では実際に使ってみましょう。下の2行はどちらも同じ意味です。

```
put the isNumber of field "tAnswer1"
put isNumber(field "tAnswer1")
```

『フィールド「tAnswer1」って数字なの？』の真偽を返しなさい。
（2行目は、randomのようにカッコを使う方法で書いています）

まず始めに、答えのフィールドが数字でなければ、そこでもうプログラミングを進める意味がないので、その時点で「mouseUpハンドラー」から抜けるように、ボタン「tAnswer1」のスクリプトを書き換えます。「isNumber」と「answer」と、もうひとつ新しいことば「exit mouseUp」を使います。「exit mouseUp」は実行をその時点で中止して、「mouseUpハンドラー」から抜ける意味です。

```
on mouseUp
   put field "tAnswer1" into tAnswer1
   if isNumber(tAnswer1) is false then
      beep
      answer "Your answer is not a number."
      select the text of field "tAnswer1"
      exit mouseUp
   end if
   put field "number1" into tNumber1
   put field "number2" into tNumber2
   if tNumber1 + tNumber2 = tAnswer1 then
      answer "Very Good! You are a Genius."
   else
      beep
      answer "Sorry, Wrong Answer."
   end if
end mouseUp
```

一番始めにフィールド「tAnswer1」の答えをバリアブル「tAnswer1」に入れます。もしそれが数字でなかったら（「if」に書かれた条件が「false　偽」だったら）ビープ音を鳴らし、アンサー・ダイアログでその内容を表示して、すぐに正しい答えに書き換えられるよう、解答欄の文字列をハイライトにセレクトして（select the text of field "tAnswer1"）、「mouseUpハンドラー」から抜けます。バリアブル「tAnswer1」が数字で、if文が「false」でなかったら、「if」の中に書かれた文は実行されないで、その後の行が実行されます。

もうひとつの問題点、もしフィールド「tAnswer1」の中でリターン・キーが叩かれたら、フィールドの1行目の文字列をそのままそっくり同じフィールドに入れる仕様にするか、すぐに答えのボタンがクリックされたようにする方法が考えられます。そっくり同じ文字列をフィールドに表示して、何も変わらないように見えるのは、下のスクリプト「returnInField」のハンドラーごと、フィールド「tAnswer1」に書き込みます。ここに書いてある「me」はオブジェクト自身の事、つまりフィールド「tAnswer1」と同じです。「returnInField」は、テキスト・フィールドのスクリプト・エディターに書かれるメッセージ・ハンドラーで、そのフィールド内でリターンが行われると「returnInField」のスクリプトが実行されます。「mouseUp」ハンドラー

と同じように「on」で始まって「end」の行で終わります。

```
on returnInField
   put line 1 of me into me
end returnInField
```

```
  put line 1 of field "tAnswer1" into feild "tAnswer1"
  （「me」を書き換えた同じ表現）
```

もうひとつの方法、答えのボタンがクリックされたようにするには、やはりフィールド「tAnswer1」の中に「returnInField」のハンドラーを書いて、 button "tAnswer1" にマウスアップ（mouseUp）を送れば、ボタンをクリックしたのと同じ結果が得られます。

```
on returnInField
   send mouseUp to button "tAnswer1"
end returnInField
```

```
  マウスアップをボタン名「tAnswer1」に送りなさい。
```

他にも対処の方法があるかもしれませんが、ここではリターンが押された後、ボタン「答えは？」にマウスアップが送られる方を使う事にします。フィールド「tAnswer1」の中に、上のスクリプトをペーストして「Apply」します。

掛け算、割り算の表記

掛け算の表記は、普通に小学校で習う算数とはちょっと違っています。例えば「2 x 4 = 8」とタイプすると、掛ける記号がアルファベットの「x」と紛らわしいと言う事がありますから「*　アスタリスク（星印）」を掛ける記号に使って、下のように表記します。

```
put 2 * 4
put 3 * 6 = 24
put 10 * (2 + 4)
```

```
  上から「8」、「false」、「60」が返されます。
```

割り算の表記も、やはり小学校の算数とは違っていて、LiveCodeでは、割り算の記号に「div」か「/」（スラッシュ）を使います。「div」は「divide（割り算する）」から来ている省略形かと思います。LiveCodeには別に「divide」と言う言葉（コマンド）もあって、少し使い方の違いがありますから、短縮形だと混同しないでください。「div」は割り算記号と考えます。「/」は分数にする記号と考えるのが良いかと思います。LiveCodeの言葉では、足し算、引き算、掛け算、割り算で使う、こう言う記号を「operator　オペレイター」と言います。

同じ割り算記号でも「div」と「/」の、二つの結果には違いがあります。「div」は、結果がプラスの場合でもマイナスの場合でも、小数点以下を切り捨てます。またはいつも整数（integer インテジャー）を返すとも言えます。「/」は小数点第7位を四捨五入して、小数点第6位までの解答をします。文章で書くと面倒そうですが、下のスクリプトを見れば、違いがすぐ分かります。

```
put 29 div 3
put 29 / 3
```
上は「9」。下は「9.666667」が返されます。

デフォルトの小数点の単位（ナンバー・フォーマット　numberFormat）は第6位ですが、いつもいつも小数点第6位までの必要もない事が多いでの、「numberFormat」で小数点第何位まで必要か、あらかじめ決めておけます。「numberFormat」のこの設定はハンドラーを抜けると、デフォルトに戻ります。

```
set the numberFormat to "0.##"
put 29 / 3
```
これで小数点第3位が四捨五入されて「9.67」が返されます。
もっと小数点以下が必要だったら、シャープ記号の数を増やします。"0.####"

もうひとつ割り算の余りを返す「mod」を書いておきましょう。「div」を使っているといつも整数が返ってくるので、時々余りが幾つか必要なことがあります。

```
put 29 mod 3
```
「2」が返されます。（余りは２）

大雑把にLiveCodeの加減乗除について書いてきました。次の章ではもう少し「算数クイズ」を発展させます。もしこの章のスタック「sansuu」を保存してなかったら、Fileメニューから「Save」してください。

Tips

- スクリプト・エディターのインデント（行の頭にある空白）がうまく表示されていないようでしたら、ハンドラーの行のどこかにカーソルを入れて「タブ・キー」を叩きます。

- 計算式は読みやすいように、カッコで囲うことができます。

この章で新しく出て来た言葉

+
オペレイター（operator）2つの数字を足す
```
put 2 + 3
put 100 + 200
```

-
オペレイター（operator）ひとつの数から他の数を引く。負の数にする
```
60 - 30
300 - (380 - 100)
```

is
オペレイター（operator）
2つの値を比べて等しいければ「true　真」、そうでなければ「false　偽」とする　同義語：=

```
(2 + 3) is 5
65 - 30 is 35
65 - 30 = 35
```

random
ファンクション（function）　無作為に選んだ整数を返す

```
put the random of 10
put random(100)
```

empty
コンスタント（constant）　何も値がない。"" は文字列に何も入っていない状態

```
put empty into field "tAnswer1"
put "" into tNumber
```

answer
コマンド（command）　ダイアログ・ボックスに、メッセージと判別を選べるボタンを出す

```
answer "Very Good!"
```

beep
コマンド（command）　システム・ビープを鳴らす

```
beep
```

isNumber
ファンクション（function）　値が数字の場合はtrue、そうでなかったらfalseを返す

```
the isNumber of field "tAnswer1"
put isNumber(field "tAnswer1")
```

exit
コントロール・ストラクチャー（control structure）　ハンドラーの途中でストップする

```
exit mouseUp
```

select
コマンド（command）　テキストをセレクトする

```
select the text of field "tAnswer1"
```

returnInField
メッセージ（message）
フィールド内でリターン・キーが叩かれた時、そのフィールドにメッセージを送る
send
コマンド（command）　メッセージをオブジェクトに送る

```
on returnInField
  send mouseUp to button "tAnswer1"
end returnInField
```

me
キーワード（keyword）　実行しているハンドラーのあるオブジェクト名に同じ

```
put line 1 of me into me
```

*
オペレイター（operator） 2つの数を掛ける
```
put 2 * 4
put 3 * 6 = 24
```
div
オペレイター（operator） ある数を他の数で割る。小数点は切り捨て、結果は整数
```
put 29 div 3
```

/
オペレイター（operator） ある数を他の数で割る。デフォルトで小数点第6位までの値を返す
```
put 29 / 3
```

numberFormat
プロパティー（property） 小数点以下の位を特定する
```
set the numberFormat to "0.##"
```

mod
オペレイター（operator） ある数を割ったあまり。この例では2
```
put 29 mod 3
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

09: 算数クイズ 2

この章の概略

- *前章の「足し算クイズ」を「足し算・引き算クイズ」に作り替えます。*
- *ビルトイン・ファンクションの説明の後、カスタム・ファンクションの作成と使用法。*
- *メッセージの伝わり方と、プライベートと言う事。オーナーとターゲットについて。*
- *カスタム・コマンドを作って、同じスクリプトで違うグループの解答を調べます。*
- *リピートで繰り返して、3問出題の問題を1回のボタン・クリックで作ります。*

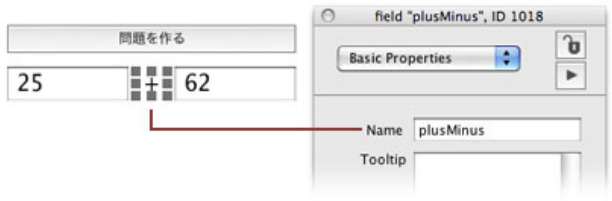
この章では、カスタム・ファンクションと、カスタム・コマンド、それとリピートの基本を学ぶのが主な目的です。LiveCodeのプログラミングは、日常英語のような簡単な言い回しで、コンピュータと会話できるコードを、順序立てて自分で組み立てられるかの訓練です。公式のように言い回しは厳密に覚える必要はなく、必要なら調べ直せば良いと思うくらいの軽い気持ちで進めてください。どんな方法でもうまく動けばそれが正解です。カスタム・ファンクションも、カスタム・コマンドも、もともとLiveCodeにビルトインされている言葉と形の上ではそう違いはなくて、それが自作できるようになっています。コードを書くのが少し慣れてきたら、小さな単位でも、まとめてファンクションやコマンドにして行くような習慣が必要かもしれません。そうして行けば、カスタム・ファンクションや、カスタム・コマンドの便利さも分かって、プロジェクト全体の流れを見ながらコードを書いて行く習慣も出てきます。

この章で使われる英単語

- date [名詞：日付]
- parameter [名詞：パラメーター, 補助変数]
- function [名詞：ファンクション, 機能, 関数]
- custom [形容詞：カスタム, あつらえの]
- private [形容詞：プライベート, 個人的な]
- definition [名詞：定義]
- message [名詞：メッセージ]
- path [名詞：道筋]
- owner [名詞：オーナー, 持ち主]
- target [名詞：ターゲット, 標的]
- hierarchy [名詞：ハイアラーキー, 階層, 階級制, ヒエラルキー]
- repeat [動詞：繰り返す]

足し算・引き算クイズ

前章のスタック「足し算クイズ」に、引き算も加えて「足し算・引き算クイズ」にします。まず始めにプラス記号のあるテキスト・フィールドの名前（Name）を「plusMinus」にしてください（下図）。算数問題を作る時にランダム（random）を使って、このテキスト・フィールドの記号を「+」または「-」に変えます。



前章で使った「足し算クイズ」のボタン「問題を作る」のスクリプトを見てみます。スクリプト中「--」の付いた日本語は説明（コメント）です。

```
-- ボタンがマウスでクリックされたら
on mouseUp
```

```
   -- 1から100までのランダムの数を、それぞれのフィールドに入れる
   put the random of 100 into field "number1"
   put the random of 100 into field "number2"

   -- 解答のフィールドは空白にする
   put empty into field "tAnswer1"

-- mouseUpハンドラーから抜ける
end mouseUp
```

このスクリプトのどこかの行に、ランダム（２）で得た数字で「プラス」か「マイナス」かを割り振って、フィールド「plusMinus」に入れれば良い訳です。その部分を赤で最後に書き加えます。

```
on mouseUp
   put the random of 100 into field "number1"
   put the random of 100 into field "number2"
   put empty into field "tAnswer1"
   get the random of 2
   if it is 1 then
      put "+" into field "plusMinus"
   else
      put "-" into field "plusMinus"
   end if
end mouseUp
```

赤文字だけの説明です。「1」か「2」の数字をランダムに発生させて、ゲット（get）で「it」の中に取り込みます。もし「it」が「1」ならばフィールド「plusMinus」にプラス記号、そうではなく（else）「2」ならばフィールド「plusMinus」にマイナス記号を入れます。1行づつ読んで行けば、そう難しい英語ではありません。これで「問題を作る」スクリプトはできました。ボタン「問題を作る（question1）」のスクリプト・エディター中に書き込みます。

今度は、前章で作った「足し算クイズ」のボタン「答えは？（tAnswer1）」のスクリプトを見てみます。スクリプト中「--」の付いた日本語の行は説明（コメント）です。

```
-- ボタンがマウスでクリックされたら
on mouseUp

   -- フィールドの答えを「tAnswer1」と言うコンテナーに入れる
   put field "tAnswer1" into tAnswer1

   -- もし「tAnswer1」が数字でなかったら、if内を実行してmouseUpから抜ける
   if isNumber(tAnswer1) is false then
      beep
      answer "Your answer is not a number."
      select the text of field "tAnswer1"
      exit mouseUp
   end if

   -- 問題のフィールドをコンテナー「tNumber1」「tNumber2」に入れる
   put field "number1" into tNumber1
```

```
   put field "number2" into tNumber2

   -- 計算式が真ならば、正解のダイアログを出す
   -- 計算式が偽ならば、ビープ音を鳴らし間違いのダイアログを出す
   if tNumber1 + tNumber2 = tAnswer1 then
      answer "Very Good! You are a Genius."
   else
      beep
      answer "Sorry, Wrong Answer."
   end if

-- mouseUpハンドラーから抜ける
end mouseUp
```

「足し算クイズ」ですから、足し算の場合だけを考えています。青文字にしているスクリプトは、プラス記号にだけ対処してあるので、「足し算・引き算クイズ」では、if文を使ってプラス記号の場合とマイナス記号の振り分けを作り、さらにそれぞれの正解、不正解のスクリプトを作ります。

if文でプラス記号の場合と、マイナス記号の場合に対処した部分だけを書きます。「--」の後の日本語は説明（コメント）ですが、スクリプトに書き込む時は削除してください。スクリプト・エディターの中に日本語は書き込めません。

```
if field "plusMinus" is "+" then
   -- プラス記号の場合（足し算クイズと同じ）
   if tNumber1 + tNumber2 = tAnswer1 then
      answer "Very Good! You are a Genius."
   else
      beep
      answer "Sorry, Wrong Answer."
   end if
else   -- プラス記号でなかったら
   -- ここにはマイナス記号の場合のスクリプトが来ます。
   if tNumber1 - tNumber2 = tAnswer1 then
      answer "Very Good! You are a Genius."
   else
      beep
      answer "Sorry, Wrong Answer."
   end if
end if
```

プラス記号の場合の青文字はそっくりそのまま前回のスクリプトです。マイナス記号の赤文字は、青文字のプラスがマイナス記号に変わっただけです。この上にあるスクリプトを、ボタン「答えは？（tAnswer1）」のマウスアップ・ハンドラーを含めて全部書きます。

```
on mouseUp
   put field "tAnswer1" into tAnswer1
   if isNumber(tAnswer1) is false then
      beep
      answer "Your answer is not a number."
      select the text of field "tAnswer1"
```

```
      exit mouseUp
   end if
   put field "number1" into tNumber1
   put field "number2" into tNumber2
   -- ここから下が変更部分
   if field "plusMinus" is "+" then  -- プラス記号の場合
      if tNumber1 + tNumber2 = tAnswer1 then
         answer "Very Good! You are a Genius."
      else
         beep
         answer "Sorry, Wrong Answer."
      end if
   else  -- プラス記号でなかったら（マイナス記号の場合）
      if tNumber1 - tNumber2 = tAnswer1 then
         answer "Very Good! You are a Genius."
      else
         beep
         answer "Sorry, Wrong Answer."
      end if
   end if
end mouseUp
```

赤文字部分には「if」で始まる文が3つあります。始めの「if」は大きく包括して「end if」が赤文字の最後の行に来ています。この「if文」は「else」で2つに分けられ、その中にもうひとつの仮定の「if文」がそれぞれ入って、やはり「else」で分けられ「end if」で終わります。どの「if」が、どの「end if」で終わっているかを確認しながらコードを書いて行くのは、とても重要です。

カスタム・ファンクションを作る

スクリプトを書き換えて「足し算・引き算クイズ」になりました。次のステップで、これを使って簡単な「カスタム・ファンクション（custom function）」と「カスタム・コマンド（custom command）」を作ります。現在のままでも機能としては働きますが、違う形に整理して、もう少しプログラミングらしさが使えるようにします。

カスタム・ファンクションは、計算したある値や、スクリプトである事を導き出した結果を返す、単体のパーツとも言える機能で、その機能から導いた結果は、カスタム・ファンクションを定義した時に付けた名前を使って、コンピューターに指令する文面（ステートメント statement）に組み込まれます。LiveCodeにはビルトイン（buit-in)・ファンクションと言うのがもともと用意してあって、これまでにも「random」や「isNumber」などのファンクションを使ってきました。ビルトイン・ファンクションは、2通りの表記ができます。

```
the date    -- 日付を取得する
the random of 100    -- 1から100までの数字を無作為にひとつ選び出す
the isNumber of field "tAnswer1"    -- フィールド「tAnswer1」が数字か判断する

date()
random(100)
isNumber(field "tAnswer1")
```

上3行のプレインな英語的言い回しでは、「the ファンクション名 of ある値」のような形を取ります。日付を取得する「the date」は「of ある値」のようなパラメータ（parameter）はありません。まったく同じ意味のもうひとつ別な形では、下3行の『ファンクション名 カッコ（ある値・状態・パラメータ)カッコ閉じる』のようにも書く事ができます。「date()」のようにパラメータ（parameter）を取らないファンクションのカッコ内は、空白になります。

カスタム・ファンクションは、ファンクション・ハンドラーと言われる形を取って、その意味の内容から返す事柄・値を決めます。これをファンクションを定義（definition）すると言っています。ファンクション・ハンドラーは、始めの行が「function」で始まり、「ファンクション名」が続き、その行の最後に必要なら「パラメータ　parameter」が来ます。その後行を変えてある何かを導き出す一連の指示の行が来て、その結果が「return」で返され、最後は必ず「end ファンクション名（始まりと同じ名前）」で終わります。この下にカスタム・ファンクションの型を書きました。パラメータ（parameter）は、上に説明したビルトイン・ファンクションのプレインな英語的言い回しの「of ある値」に相当する、ファンクションを実行する際に指定で変わる値です。パラメータがないカスタム・ファンクションも、幾つも使うカスタム・ファンクションもあります。

```
function ファンクション名 パラメータ1, パラメータ2
  ある事を導き出す指示とそのプロセスのステートメントの行
  return プロセスの結果
end ファンクション名
```

具体的な文面を見た方が分かりやすいでしょう。では「足し算・引き算クイズ」のボタン「問題を作る（question1）」に、この章で作った新しいスクリプト部分を「plusOrMinus」と言う名前のカスタム・ファンクションにします。このファンクションにはパラメータはありません。赤文字の行がカスタム・ファンクションの定義です。

```
function plusOrMinus
   get the random of 2
   if it is 1 then
      return "+"
   else
      return "-"
   end if
end plusOrMinus

get the random of 2
if it is 1 then
      put "+" into field "plusMinus"
   else
      put "-" into field "plusMinus"
end if
```

緑文字で書いてあるのは、上で作った「足し算・引き算クイズ」で使ったスクリプトです。違いはカスタム・ファンクションでは、始めの行でこれはファンクションであると言う「function」＋「plusOrMinus （ファンクション名）」が入っている事と、「retrun」でプラス記号かマイナス記号を返しています。そうでない方は、フィールドの中に直接記号を入れてい

ます。

カスタム・ファンクションをステートメントに組み込むには、「plusOrMinus()」（ファンクション名 + カッコ）にします。ビルトイン・ファンクションの「of ある値」を使った、英語表現はできません。ステートメントの中でカスタム・ファンクションを使うのを、LiveCodeではファンクションを呼び出す（Call）と呼んでいます。下の例ではカスタム・ファンクションを一カ所だけで使っていますが、カスタム・ファンクションの利点は、短い言葉で同じ処理の情報を何カ所にも使うことができたり、定義の修正だけで、呼び出されたファンクション全てを一度に修正ができる利点があります。

以下が、ボタン「問題を作る（question1）」に入れる、カスタム・ファンクションを使った全部のスクリプトです。「--」の付いている日本語は説明（コメント）ですから、スクリプトで使う時は削除してください。スクリプト・エディターに日本語は書き込めません。

```
on mouseUp
   put the random of 100 into field "number1"
   put the random of 100 into field "number2"
   put empty into field "tAnswer1"

   -- ファンクション名() で呼び出し「1」または「2」が返される
   put plusOrMinus() into field "plusMinus"
end mouseUp

function plusOrMinus
   get random(2)
   if it is 1 then
      return "+"    -- 呼び出された時プラス記号を返す
   else
      return "-"    -- 呼び出された時マイナス記号を返す
   end if
end plusOrMinus
```

図1：エディター（左にオブジェクト内のハンドラーの一覧、右は書かれたスクリプト）

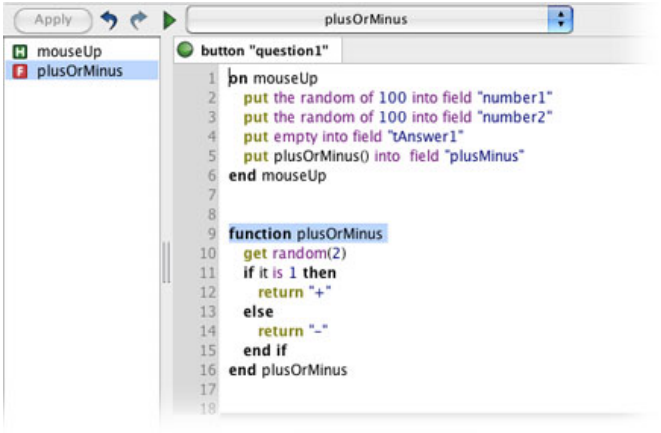
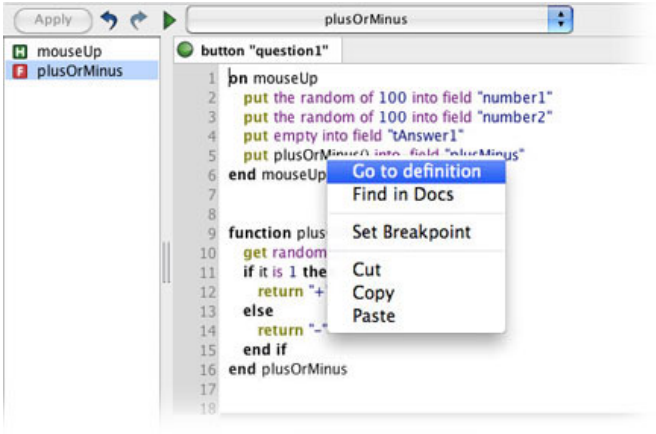


図2：呼び出しているファンクション名からコンテキスト・メニューを出す



上図はどちらもボタン「問題を作る（question1）」のスクリプト・エディターです。図1は、左のハンドラーの一覧からハンドラー名をセレクトすると、右にあるそのハンドラー名が選ばれます。図2は呼び出しているファンクションからコンテクスト・メニューが出ますから、そのファンクションの定義（definition）がどこにあるかを見つける時に使います。ここでは一つだけのファンクションですが、数が増えて来ると非常に便利です。

この例では定義（definition）しているファンクションと、呼び出されて使われているファンクション（ファンクション名＋カッコ）とが、同じオブジェクトの中にあります。幾つものオブジェクトから、ひとつのファンクションを呼び出すようにするには、マウスアップ等のアクションが、メッセージとしてLiveCodeのエンジンに伝わって行くルートの上位に、ファンクションの定義（definition）が置かれていなければいけません。LiveCodeは、もしオブジェクト内にファンクション・ハンドラーが見あたらなかったら、メッセージが伝わるルート（メッセージ・パス）の上位に、ファンクションの定義を探しに行きます。もしカード上の幾つかのボタンから「plusOrMinus()ファンクション」が呼び出されるとすると、この「plusOrMinus」ファンクションの定義は、ボタンが置いてあるカードのスクリプト・エディターの中か、さらに上位のスタックのスクリプト・エディターの中に書かれていなければいけません。このメッセージをエンジンまで伝える階層のルートを、メッセージ・パス（message path）と呼んでいます。

プライベート「private」

グローバル（global）とローカル（local）について話した事と少し似ていて、呼び出されるファンクションが、同じオブジェクト内（ローカル）で定義されているのを、LiveCodeではプライベート（private）と言って、「function」の前にキーワードの「private」を付ける事ができます。これは必ず付けなければいけない言葉ではありません。プライベート（private）を付ける事によって、LiveCodeエンジンが上位階層にカスタム・ファンクションを探しに行く必要がなくなって、それだけ処理スピードを早めます。

```
private function plusOrMinus
   get random(2)
   if it is 1 then
      return "+"
   else
      return "-"
```

```
   end if
end plusOrMinus
```

プライベート（private）は同じオブジェクト内だけで使えるキーワードですから、メッセージの伝わる別な階層で使うとエラーになります。 プライベート（private）はカスタム・ファンクション（custom function）だけでなく、これから学ぶカスタム・コマンド（custom command）でも使われます。

カスタム・コマンドを作る

これもファンクションのように、LiveCode自体に用意されている「create」set」「put」等のようなビルトイン・コマンドがあります。カスタム・コマンド（custom command）は、日本語で言う「命令語」をカスタムで作ります。ビルトイン・コマンドのような、日常英語のように使える柔軟性はあまりませんが、長い記述のコマンドを、短い言葉に短縮して何度も使う事ができます。LiveCodeに類似した、日常英語のような他のコンピュータ言語（HyperTalk、SuperTalk等）では

```
on コマンド名 パラメータ1, パラメータ2
  -- カスタムで作る命令語のステートメントがここに来る
end コマンド名
```

のように、「on + コマンド名」で始まり、必要によって「パラメータ」、行を変えて「命令語のステートメント」、最後は「end + コマンド名」で終わります。LiveCodeでもこの伝統的な形のカスタム・コマンドは使えますが、前述のプライベート（private）を使う事もあって、下のようにコマンドを作ります。

同じオブジェクト内にプライベートでカスタム・コマンドを定義する場合

```
private command コマンド名 パラメータ1, パラメータ2
  -- カスタムで作る命令語のステートメントがここに来る
end コマンド名
```

メッセージ・パス（message path）の上位に置く場合

```
command コマンド名 パラメータ1, パラメータ2
  -- カスタムで作る命令語のステートメントがここに来る
end コマンド名
```

カスタム・コマンドを使う場合は、

```
コマンド名, パラメータ1, パラメータ2
```

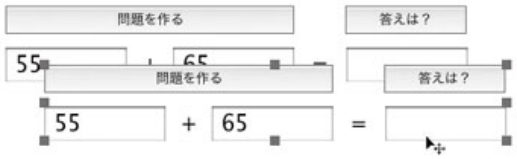
のように完結する独立した行で記述して、ビルトイン・コマンドのように、「to」や「into」などのキーワードと一緒に、ステートメントの中の言葉として使う事はできません。

それでは「足し算・引き算クイズ」でカスタム・コマンドと、プライベートでないカスタム・ファンクションを作ってみます。カード上にボタン「答えは？」を3つ作って、それぞれのボタンから、カスタム・コマンドを使います。このカスタム・ファンクションとカスタム・コマンドの定義は、必ずカードより上位のメッセージ・パスになければいけません。最後にひとつだけの「問題を作る」ボタンで、3問出題できる「足し算・引き算問題」のスクリプトを作りま

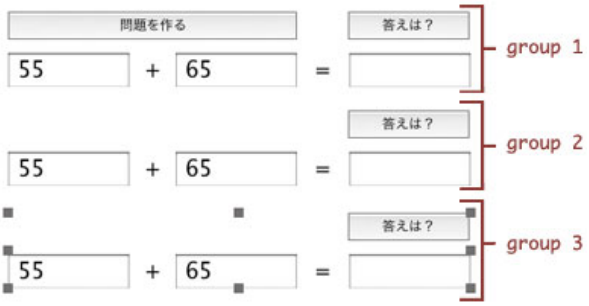
す。

始めに今カード上にあるボタン2つと、フィールド5つを纏めてセレクトしたら、「Objectメニュー＞Group Selcted」を選んで「group 1（グループ１）」を作ります。「グループ１」ができたら、Mac OSはオプション・キー（Windowsはオルト・キー）を押しながら、「グループ１」をドラッグしてコピーの「グループ2」と「グループ3」を作ってください。「グループ１」のボタン「問題を作る（question1）」だけ残して、「グループ2」と「グループ3」のボタン「問題を作る（question1）」は削除します。削除はアプリケーション・ブラウザーからでも、アイコン・メニューの「Edit Group」からでも、どちらからでもできます。グループのナンバーは、階層の下のレイヤーから1,2,3と振られて行きます。最後に作ったグループは、ナンバー「3」です。レイアウト上ナンバーを入れ替えたいときは、グループのインスペクターの「Size & Position」でナンバー（Number）を変えられます。

図：グループのコピー



図：グループ１、グループ２、グループ３



カスタム・コマンドを使う「答えは？」の方から作って行く事にしましょう。どのグループでも良いですからボタン「答えは？ tAnswer1」のスクリプト・エディターを開きます。ボタンはグループ内にあるので、エディット・ツールでグループをセレクトしてアイコン・メニューの「Edit Group」をクリックして、ボタンをセレクトするか、ブラウズ・ツールでMacOSなら「オプション＋コマンド・キー」、Windowsでは「コントロール＋オルト・キー」でターゲットの オブジェクトをクリックすれば、スクリプト・エディターが開きます。

エディターを開いたら、マウスアップのスクリプトを「on mouseUp」「end mouseUp」だけ残して全てカットして、カードのスクリプト・エディターにペーストします。カードのスクリプト・エディターはカード上のどこでもマウスで、MacOSなら「コマンド＋オプション＋シフト・キー」、Windowsなら「シフト＋コントロール＋オルト・キー」でクリックすると、コンテキスト・メニューが出ますから、「Edit Card Script」を選びます。または「Objectメニュー＞Card Script」を選びます。

ボタン「答えは？ tAnswer1」からカットしたスクリプト（上図）をカードのスクリプト・エディターにペーストしたら、始めに「command tAnswer pGroupNum」の行を書き入れ、最後には「end tAnswer」の行を書き足します。このカスタム・コマンドの名前は「tAnswer」です。このカードの上にはグループのレイヤーが3つあって、同じ名前のテキスト・フィールドもそれぞれのグループにあります。スクリプトから何かの指示を、同じ名前のオブジェクトに出した場合、最下位のレイヤーにあるオブジェクトと判断されます。それを避けるにはどのグループに、そのオブジェクト（この場合はフィールド）が属しているかがキーとなります。この例ではグループ・ナンバーをパラメータ（parameter）として使い分ける事にしましょう。

カードにある「答えは？」の3つのボタンから送るコマンドは、それぞれグループ・ナンバーを差し替えて、カード内にあるカスタム・コマンド「tAnswer」の定義を読み取ると言う事になります。一般的にLiveCodeディベロッパーは、パラメータの定義でどの言葉がパラメータなのか分かるように、頭に「p」の文字を付けて使うことが多いので、それに倣ってここでも「pGroupNum」と言うパラメータにしました。「グループ１のボタン」から送られるときは「1」、「グループ2のボタン」から送られるときは「2」と、差し替えてコンピュータに読まれる事となります。

変更部分に1行だけグループ・ナンバーではない行があります。ボタンの中でマウスアップ・ハンドラーに書かれていた「exit mouseUp」は、ハンドラーのマウスアップから抜ける指示でしたが、カスタム・コマンドのハンドラーは「command tAnswer」になっているので「exit mouseUp」を使うとエラーになってしまいます。この行は「exit to top」にし変更ます。「exit to top」は、現在のハンドラー（command tAnswer）も、他の経過中のハンドラー（mouseUp）も、すべての実行を中断させます。カードにペーストしたスクリプトに、新しく書き加えられた部分を赤文字にしています。

```
command tAnswer pGroupNum
   put field "tAnswer1" of group pGroupNum into tAnswer1
   if isNumber(tAnswer1) is false then
      beep
      answer "Your answer is not a number."
      select the text of field "tAnswer1" of group pGroupNum
```

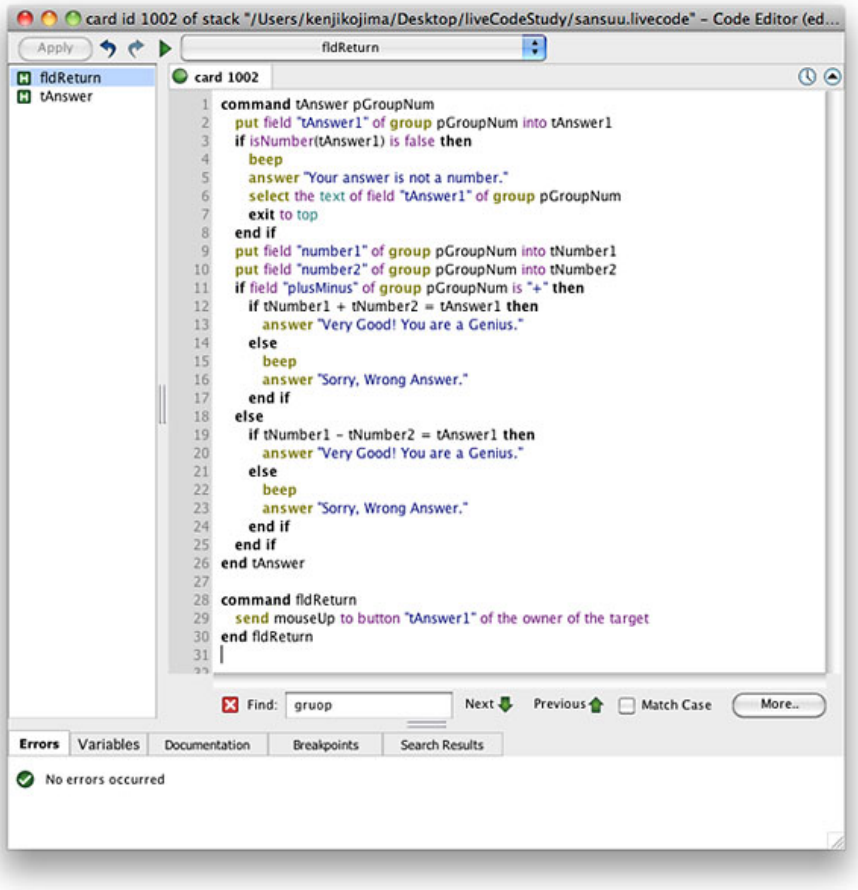
```
      exit to top    -- この行だけグループ・ナンバーではない変更
   end if
   put field "number1" of group pGroupNum into tNumber1
   put field "number2" of group pGroupNum into tNumber2
   if field "plusMinus" of group pGroupNum is "+" then
      if tNumber1 + tNumber2 = tAnswer1 then
         answer "Very Good! You are a Genius."
      else
         beep
         answer "Sorry, Wrong Answer."
      end if
   else
      if tNumber1 - tNumber2 = tAnswer1 then
         answer "Very Good! You are a Genius."
      else
         beep
         answer "Sorry, Wrong Answer."
      end if
   end if
end tAnswer
```

実はもうひとつ解答するフィールドでリターンがされた時、「答えは？」のボタンに「mouseUp」を送るカスタム・コマンドが必要です。こちらは「fldReturn」と言う名前のカスタム・コマンドで、パラメータはありません。パラメータの変わりに別な新しい言葉「target」と「owner」を使う事にします。

```
command fldReturn
   send mouseUp to button "tAnswer1" of the owner of the target
end fldReturn
```

「target」は、クリック等のアクションが行われたオブジェクト名を指します。別な表現では、「メッセージの実行のスタートが受け取られたオブジェクト名を返す」と言う内容になります。もうひとつの「owner」のこのスクリプトでの意味は、オブジェクトが属しているグループ名です。LiveCodeの言葉としての意味では、ひとつ、ひとつのオブジェクトには、それぞれオーナーが決まっていて、コントロール、グループ、カード、スタックと言う階層の「object hierarchy　オブジェクト・ハイアラキー」で使われる、ひとつ上位の階層を「owner　オーナー」と言っています。オブジェクト・ハイアラキーとは、上位のオブジェクトのカラーやフォント等のプロパティに、下位のオブジェクトが従う決まりがあります。これについては、そういう事柄があるというだけにここでは留めます。

図：2つのコマンドの定義が書かれた、カードのスクリプト・エディター。




```
on mouseUp
   tAnswer 1
end mouseUp
```

グループ１のボタン「答えは？」に入れるスクリプト

```
on returnInField
   fldReturn
end returnInField
```

グループ１のフィールド「**tAnswer1**」に入れるスクリプト

```
on mouseUp
   tAnswer 2
end mouseUp
```

グループ2のボタン「答えは？」に入れるスクリプト

```
on returnInField
   fldReturn
end returnInField
```

グループ2のフィールド「**tAnswer1**」に入れるスクリプト

```
on mouseUp
   tAnswer 3
end mouseUp
```

グループ3のボタン「答えは？」に入れるスクリプト

```
on returnInField
   fldReturn
end returnInField
```

グループ3のフィールド「tAnswer1」に入れるスクリプト

以上をグループ内のそれぞれのボタン「答えは？」と解答欄のフィールドに入れます。

リピート「repeat」でループを作る

最後に、出題を一度に3問作り出すボタン「問題を作る」に取りかかりましょう。これは「足し算・引き算クイズ」で使った出題と、ほぼ同じスクリプトを3回続けて使って、毎回違うグループのテキスト・フィールドに、数字とプラス・マイナス記号を入れて行きます。これには「repeat リピート」と言う、同じステートメントを繰り返してループを作り出す、コントロール・ストラクチャーを用います。簡単なリピートの説明から始めます。「add」は数字の入っているバリアブル（下の例では「tNumber」。仮に値を入れておくコンテナー）に数字を足して、値を変えるコマンドです。

```
put 10 into tNumber  -- tNumberと言うコンテナーに10を入れる
repeat 3 times   -- 同じステートメントを3回繰り返す（ループ）
  add 2 to tNumber   -- tNumberに2を加える
end repeat   -- 繰り返し終わり
put tNumber    -- tNumberをメッセージ・ボックスに
```

10 + 2 x 3 と同じ結果になります。

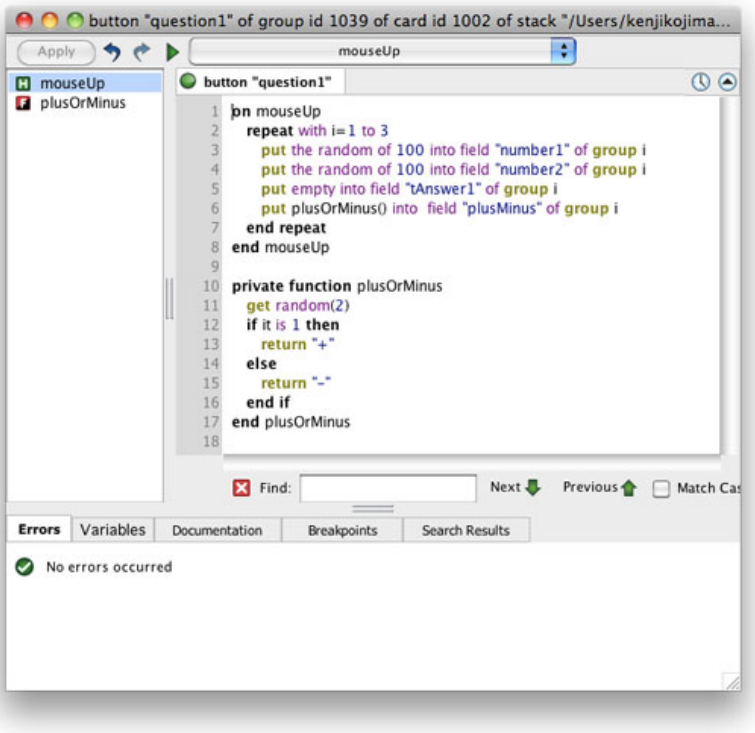
リピートを構成する基本的な形は「repeat 」＋「ループの回数指示（ループフォルム LoopForm)」、行を変えて「どういう作業を行うかのリスト」に従って、指定された回数だけ作業を行い、最後に「end repat」でループから抜けます。上のループの回数指示（ループフォルム　LoopForm）は「3 times」でしたが、これ以外にも「forever　（永久に）」とか「until 何々になった場合（コンディション）」などのループフォルムがあります。

ボタン「問題を作る」では「with カウンターに使う文字 = 始めの数字 to 終わりの数字」と言うループフォルムを使います。こういう場合の「ループ・カウンター」によく使われる文字の「i」にしています。これは「integer 整数」の頭文字をループ・カウンターとして使う慣用的な用法で、「X」でも「j」でもスクリプト中で支障のない文字であれば何でもかまいません。

```
repeat with i=1 to 3   -- i の数字がループで1から3まで変わります
  -- ループの都度グループ・ナンバーの数字 i が変わって行きます。
  put the random of 100 into field "number1" of group i
  put the random of 100 into field "number2" of group i
  put empty into field "tAnswer1" of group i
  put plusOrMinus() into  field "plusMinus" of group i
end repeat
```

青文字は出題がひとつだけのボタン「問題を作る」にあったスクリプトと同じです。3問出題のボタン「問題を作る」のスクリプト・エディターは下のようになります。ファンクション「plusOrMinus」は、ボタン「question1 問題を作る」の中だけで呼び出されるので、定義に「private」を付けています。

図：ボタン「question1 問題を作る」のスクリプト・エディター



Tips

- 「if」を使った仮定文に、入れ子状態でまた別な「if文」を組み込んで行く場合、それぞれの「if」がどの「else」で仮定が分けられ、どの「end if」で終わるか、いつも確認しながら書き進めます。

- スクリプト・エディターの左のコラムに出るハンドラー名の一覧から、目的のハンドラーをクリックすると、右のステートメントからその定義を探し出す事ができます。

- スクリプト・エディターの右のステートメントで呼び出されているファンクション名、使われているコマンド名から、コンテクスト・メニューを引き出すと、その定義を見つける事ができます。

- 同じオブジェクト内にあるファンクションに呼び出される、定義しているファンクション・ハンドラーの始めに「privateキーワード」を付けると、処理のスピードが早くなります。

- ファンクションの定義は、呼び出すファンクションと同じオブジェクト内か、メッセージ・パス（message path）の上位に置かなくてはいけません。

- 一般的にLiveCodeディベロッパーは、パラメータ定義でどの言葉がパラメータなのか分かるように、頭に「p」の文字を付けて使う事が多いです。

- オブジェクト・ハイアラキー（object hierarchy）によって、上位のオブジェクトのカラーやフォント等のプロパティに、下位のオブジェクトが従う決まりがあります。

- カードのスクリプト・エディターは「Objectメニュー」から「Card Script」を選ぶか、カード上どこでも右クリックして、コンテキスト・メニューから「Edit Card Script」を選びます。

この章で新しく出て来た言葉

date
ファンクッション（function）日付けを返す
```
the date
date()
```

function
コントロール・ストラクチャー（control structure）
カスタム・ファンクション　ハンドラーを定義する
```
function plusOrMinus
```

return
コントロール・ストラクチャー（control structure）
ハンドラーからある値を返す
```
return someValue
```

private
キーワード（keyword）
ファンクションかコマンドの頭に付いてオブジェクト内に定義があるかを示す
```
private function myFunction
private command myCommand
```

owner
プロパティ（property）オブジェクトのひとつ上の階層を指し示す
```
the owner of buttom 1
```

target
ファンクション（function）メッセージがスタートした時点のオブジェクトの名前を返す
```
the target
```

command
コントロール・ストラクチャー（control structure）
コマンドを定義するメッセージ・ハンドラー　同義語：on
```
command myCommand pNumber
```

add
コマンド（command）あるコンテナー（variable）に入っている数字に、別な数字を加える
```
add 3 to tNumber
```

repeat
コントロール・ストラクチャー（control structure）あるステートメントを繰り返す
```
repeat 3 times
repeat with i=1 to 5
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

10: テキスト・フィールド

この章の概略

- *スタックがリサイズされた時に、オブジェクトのサイズやポジションを変えるジオメトリー・マネージャーで、フィールド、ボタンが1つづつと、それぞれ2つづつの場合の設定について学びます。*
- *フィールドからキャラクター、ワード、ライン等のデータを得て、他のフィードに移し替えたり、フォントやカラーを変化させたりします。*
- *テキストがロックされているフィールドを、マウス・クリックでキャラクター、ワード、ライン等の情報を得ます。*
- *英語のテキストを保存します。*
- *ジオメトリー・マネージャーを使わない、スクリプトによるリサイズ・スタック。*

この章で使われる英単語

- geometry [名詞：幾何学]
- scroll [動詞：スクロールする, 巻く]
- resize [動詞：リサイズする, 大きさを変える]
- replace [動詞：入れ替わる]
- vertical [形容詞：垂直の]
- sort [名詞, 動詞：ソート（する）, 並べ替え（る）]
- chunk [名詞：かたまり]

ジオメトリー：フィールド1つ、ボタン1つ。

始めにスタックとフィールドのビジュアルな関連から入ります。スタックのサイズが変化した時に、フィールドも同時にそれに合わせて大きさが変わる、ジオメトリー・マネージャーの扱いです。ニュー・メイン・スタックをFileメニューから作ります。スタック・ネームは「textField」、タイトルは「Text Field」にします。ツール・パレットからテキスト・フィールドの「Scrolling Field」をカードにドラッグ&ドロップしてください。これは普通にテキストが打ち込めて、右側に縦のスクローリング・バー（vScrollbar）があるフィールドです。

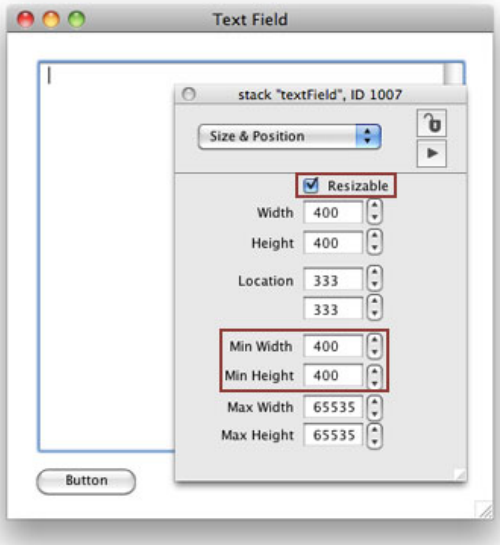
```
put the vScrollbar of field 1
```

プロパティ「vScrollbar」が「true」です。「v」は「vertical（垂直）」の意味

```
set the vScrollBar of field 1 to false
```

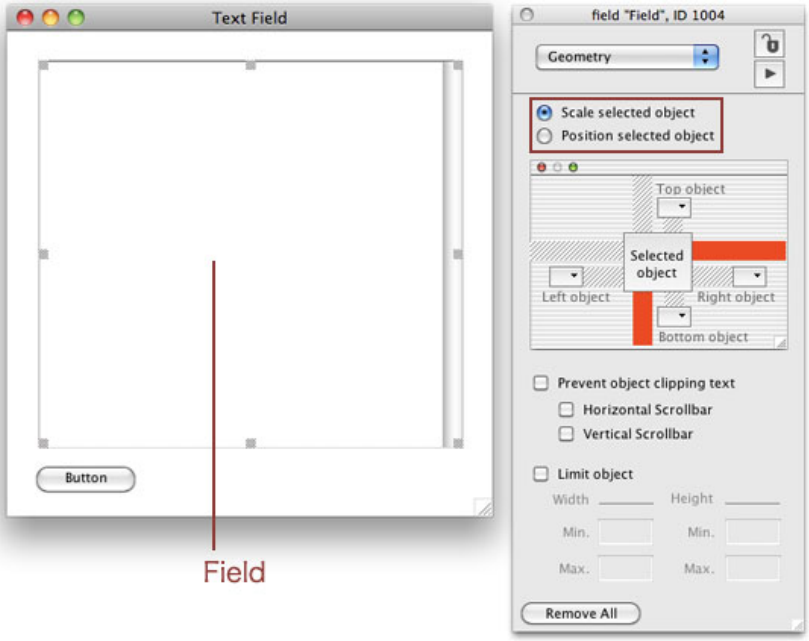
プロパティ「vScrollbar」を「false」にすると、スクロールバーが消えます。

フィールドを下の図のようにカード上に広げます。左下にボタンを置くので、そのスペースも少し見てください。左下に「Push Button」を置きます。スタック「textField」のインスペクターを開いて、「Size」& Position」を開けます。フィールドやボタン等コントロールの「Size」& Position」とは違っていて、一番上のプロパティが「Resizable（リサイズできる）」になっています。デフォルトのスタック・サイズは「400 x 400」です。下の方にある「Min Width」と「Min Height」はどちらも「400」にしてください。ユーザーがスタック（ウインドウ）のサイズを変えられる最小の大きさです。つまりこのサイズ以下にはできません。



次に「フィールド１」をセレクトして、インスペクターの「Geometry」を開きます。これでユーザーがスタック右下をドラッグして、ウインドウ・サイズが変わった時に「フィールド１」も同時に大きさが変わるようにセットします。インスペクターのトップにある「Scale selected object　セレクトしているオブジェクトのサイズを変える」が選ばれているのを確認します。その下にあるウインドウの図の中央「Selected object」から右サイドまで伸びている斜線をクリックすると、下の図のようにオレンジに変わります。同じように「Selected object」から下に伸びている斜線をクリックしてください。やはり下の図のようにオレンジに変わります。これでウインドウ・サイズに比例してフィールドのサイズも変わります。

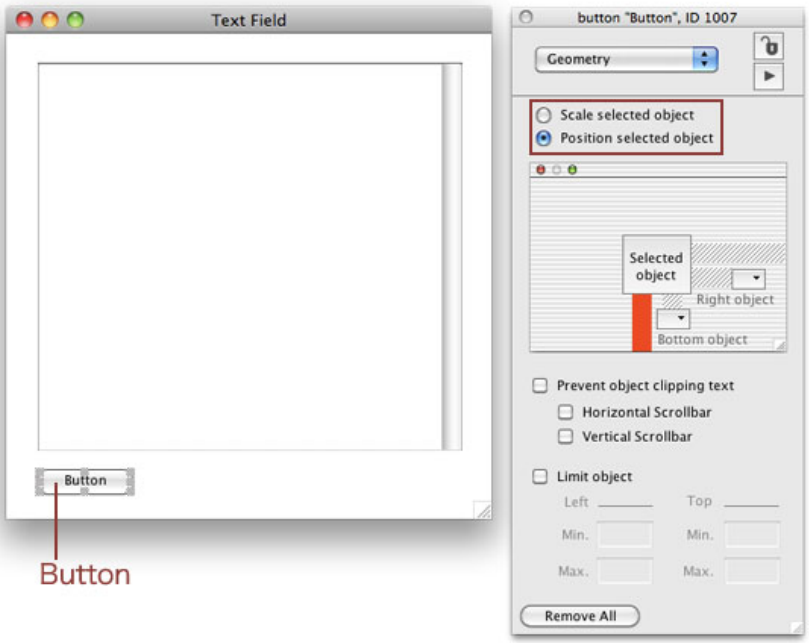
図：フィールドのジオメトリー設定



ウインドウとフィールドに合わせて、ボタンの位置が動くようにしないといけません。ボタンをセレクトして、ボタンのインスペクターから「Geometry」を開きます。今度はサイズは変わらないで、ポジションだけが移動するようにするので、トップの「Position selected object」を

選びます。ウインドウの図の中にある「Selected object」から、下に伸びている斜線をクリックしてください。下の図のようにオレンジに変わります。オレンジ色のベタ帯は、いつも変わらないピクセル数のアキを示しています。

図：左下ボタンのジオメトリー設定



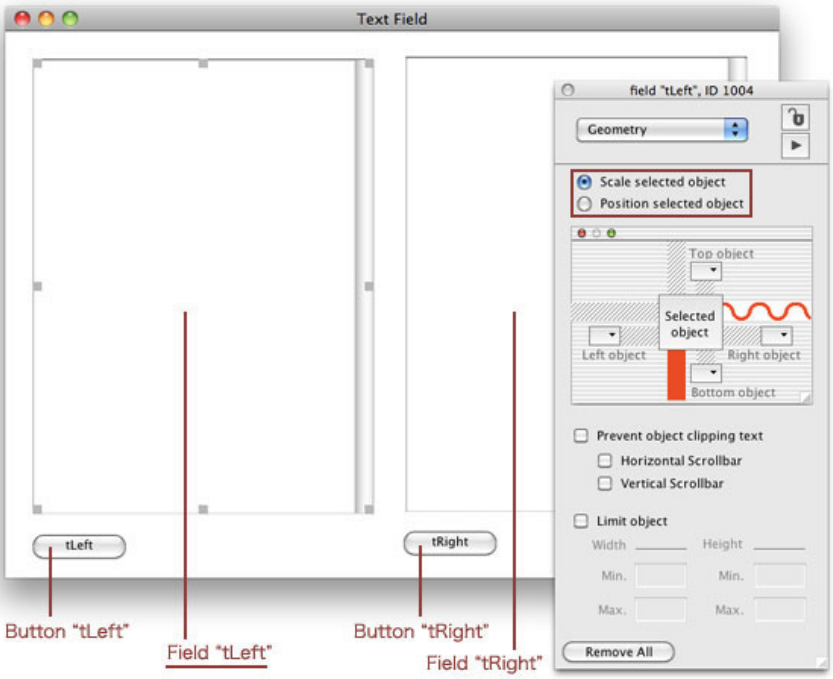
これでブラウズ・ツールに持ち替えたら、スタック右下を動かしてウインドウの大きさを変えてみましょう。どうでしょう。うまくリサイズしていますか。

ジオメトリー：フィールド2つ、ボタン2つ。

今度はフィールド2つと、ボタン2つでやってみます。いったんフィールドもボタンも「Geometry」を全て解除してください。オブジェクトをセレクトして、インスペクターの「Geometry」の最下部にある「Remove All」をクリックします。

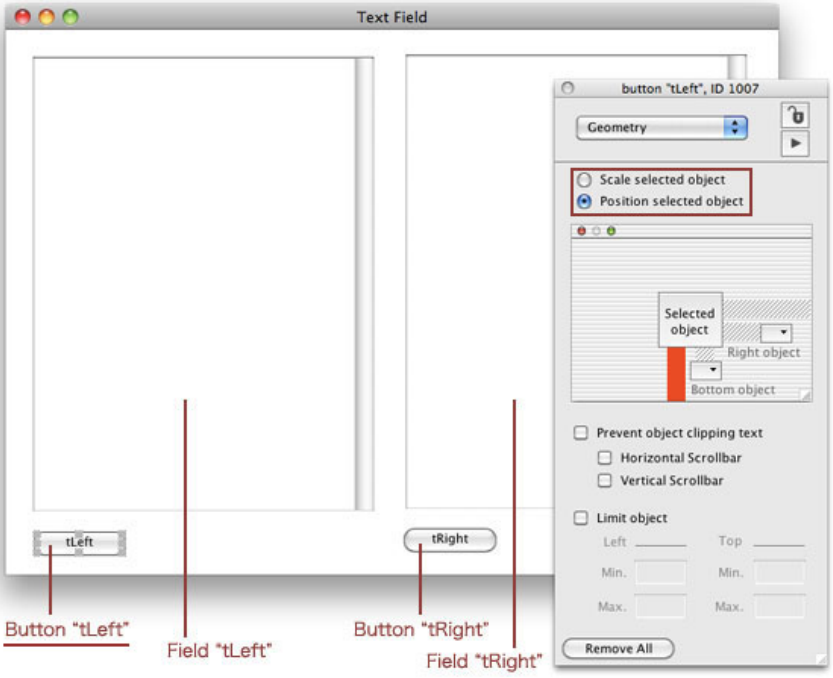
スタックをもう少し横長にします。例では 680 x 480 のサイズにしました。スタック・インスペクターで「Min Width」も「Min Height」680, 480 にしてください。フィールド 1 の名前を「tLeft」にして、をスタックの横巾のほぼ半分弱くらいのサイズにして、縦もボタンの位置ががちょうど良いくらいに伸ばします。フィールド「tLeft」の下にあるボタンは「tLeft」と言う名前にします。カードの右側の空間に同じ大きさのフィールドとボタンを作ったら、フィールド名「tRight」、ボタン名「tRight」にします。

図：左のフィールドのジオメトリー設定



始めに左のフィールド「tLeft」をセレクトして、インスペクターの「Geometry」を開いたら、トップの「Scale selected object」を選びます。ウインドウの図の中央「Selected object」から右サイドまで伸びている斜線を2度クリックすると、1度目はオレンジ色のベタの帯、2度目にオレンジの波のような線に変わります。たぶんスプリングとかゴムのような伸び縮みするイメージで、インターフェイスが作られたかと思います。「Selected object」の下にある斜線は1度だけのクリックで、オレンジのベタにします。

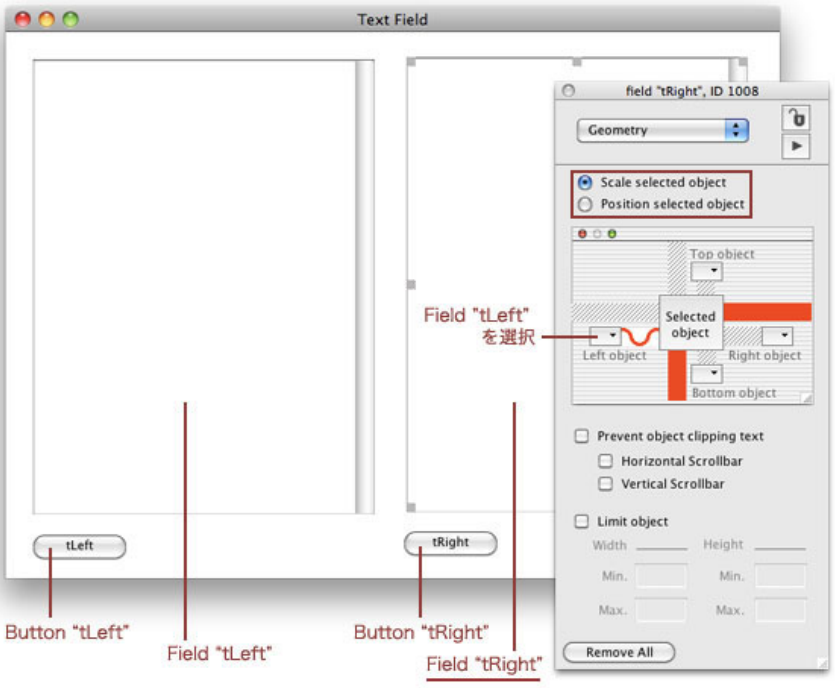
図：左のボタンのジオメトリー設定



次に左下のボタン「tLeft」は、フィールドがひとつだけの場合と同じ設定で、トップの「Position selected object」を選んで、ウインドウの図にある、「Selected object」から下に伸

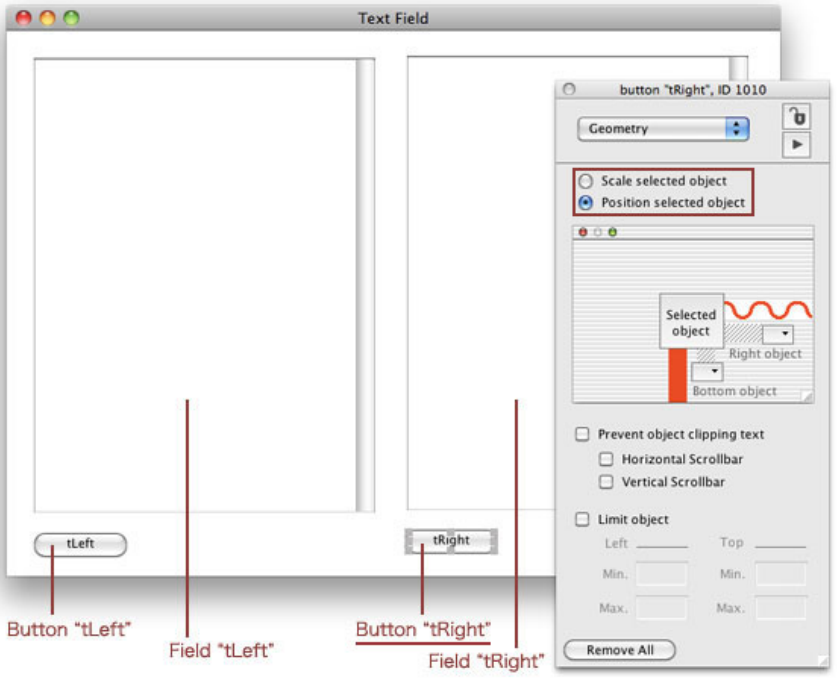
びている斜線は1度だけクリックして図のようにオレンジのベタにします。

図：右のフィールドのジオメトリー設定



右のフィールドは、トップの「Scale selected object」を選んで、右の辺、下の辺はオレンジのベタ、左は「Left object」から「Field "tLeft"」を選びます。これによって左のフィールド「tLeft」伸び縮みによってフィールド「tRight」の左サイドの位置が変わります。

図：右のボタンのジオメトリー設定

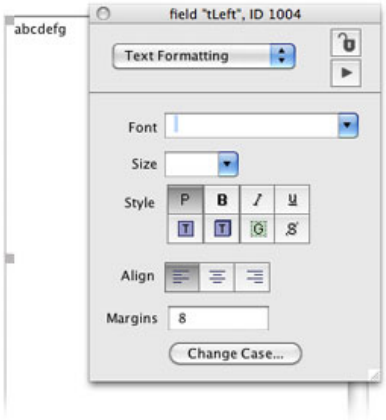


もうひとつ残っている右下のボタン「tRight」は、トップの「Position selected object」を選んで、ウインドウの下ボトムに伸びる線はオレンジ・ベタ、右サイドに伸びる線はスプリング（波線）にします。これでスタックを保存（Save）します。スタックをリサイズして、2つの

フィールドとボタンがうまく動かないに時は、全てのコントロールを「Remove All」で設定を外して、始めからやり直してください。

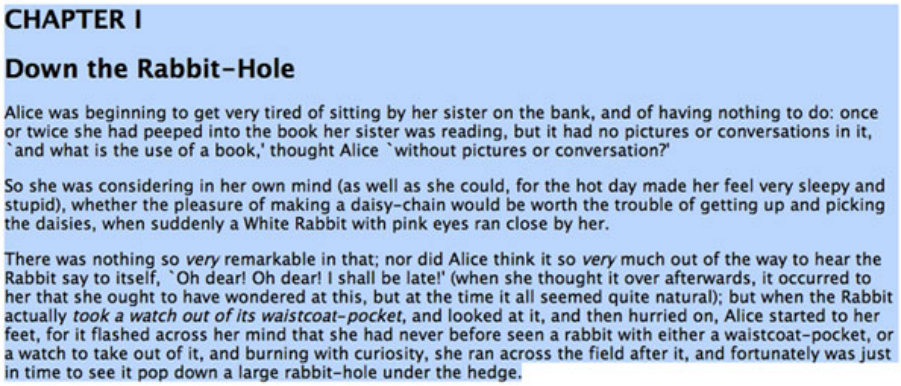
フィールドの英文を扱う

上で設定したテキスト・フィールドを使って、英文を少し触ってみます。日本文は英文がある程度扱えるくらいの、基礎的なフィールドの知識をつけてからにします。作った普通（プロパティがデフォルト）のフィールドに、「abcdefg』とタイプしてください。フィールドのインスペクターを見ると「Font　フォント」「Size　サイズ」が空白です。これはOSのシステム・フォントで、Mac OSでは「Lucinda Grande」、Windowsでは「Segoe UI」が設定されています。フォント・サイズは「11ポイント」です。



フィールドのサンプルで扱える英文をインターネットからコピーして使う事にします。「http://kenjikojima.com/livecode/alice.html」を開いてください。不思議な国のアリスの第一章の一部があります。これをサンプル・テキストとして始めの「CHAPTER I」からパラグラフ3の終わり「a large rabbit[hole under the hedge.」までをコピーして、上で作ったフィールド「tLeft」にペーストします。

図：http://kenjikojima.com/livecode/alice.htmlからコピー



CHAPTER I

Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, `and what is the use of a book,' thought Alice `without pictures or conversation?'

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so *very* remarkable in that; nor did Alice think it so *very* much out of the way to hear the Rabbit say to itself, `Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually *took a watch out of its waistcoat-pocket*, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.

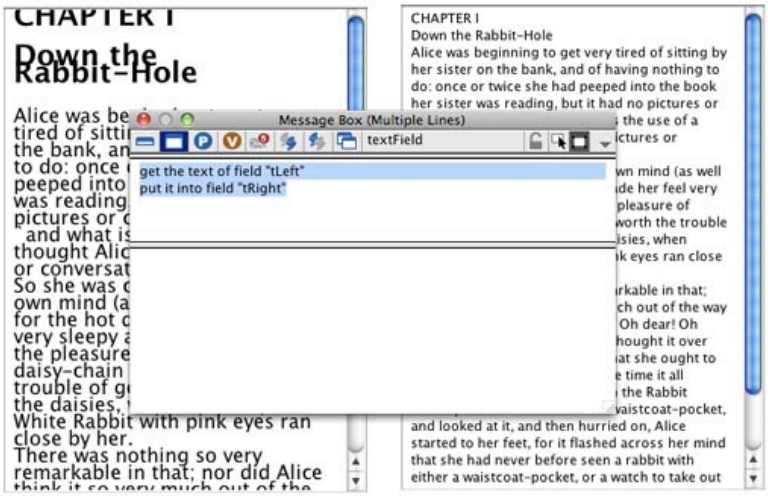
図：フィールド「tLeft」にペースト

ウェブ・ブラウザーからフィールドにペーストすると、コピーした元のページのタグ指定がそのまま生きて（LiveCodeフィールドのhtmlText設定）、フォントのサイズがフィールド・プロパティの設定ではない大きさになります。それではタグ指定を外したプレインなテキストを、フィールド「tRight」に入れます。

```
get the text of field "tLeft"
put it into field "tRight"
```

フィールド「tLeft」をゲットして「it」に入れる。
「it」をフィールド「tRight」に入れなさい。
-- put field "tLeft" into field "tRight" としても同じ結果が得れます。

図：タグの設定を取ってフィールド「tRight」に入れる

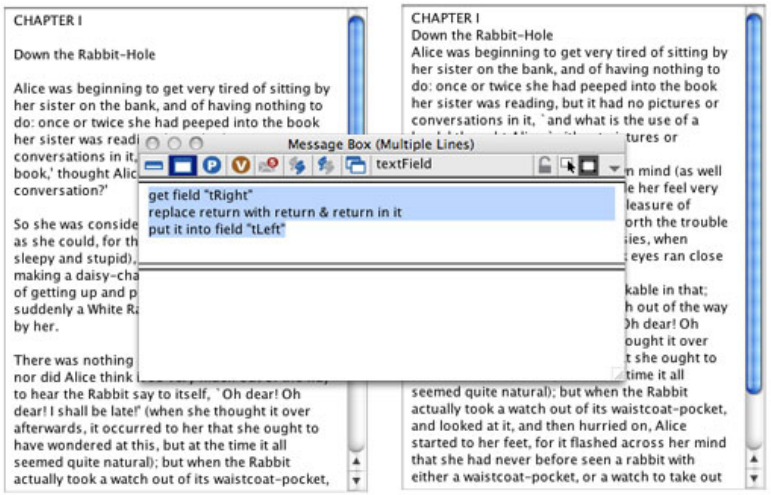


フィールド「tRight」にあるテキストをは、HTMLのタグを取り除いてフィールド・プロパティのテキストにしたので、HTMLのパラグラフのアキだった「<P>」もなくなって、タグがあったその部分のアキの行が、フィールド「tRight」からもなくなってしまいました。アキの行をメッセージ・ボックスから追加しておきます。例えば始めの１行「CHAPTER I」の最後には見えない「return　リターンのキャラクター」が入って、次の行「Down the ...」が始まっています。「return」のある全ての箇所に、もうひとつ「return」を追加して入れます。これにはコマンドの「replace（差し替え）」を使う事にします。フィールド「tRight」をゲットで受け、「it」の中にある全ての「return」キャラクターを「return & return」に差し替えたら、その「it」をフィールド「tLeft」に入れます。

```
get field "tRight"
replace return with return & return in it
put it into field "tLeft"
```

フィールド「tRight」をゲットして「it」に入れる
「it」の中の「return」キャラクターを「return & return」に差し替えなさい。
「it」をフィールド「tLeft」に入れなさい。
-- 「return」キャラクターは「cr」と同じ意味でなので「cr & cr」とも書けます。
-- 「cr」は Carriage Return の略。

図：「return」キャラクター を「retrun & return」に差し替え



それではキャラクターやライン等、テキストの基本的な事を調べて行きます。 始めに何文字のキャラクターがフィールド「tLeft」にあるか、尋ねてみます。キャラクターは複数形「characters」を使います。

```
put the number of characters of field "tLeft" into field "tRight"
```

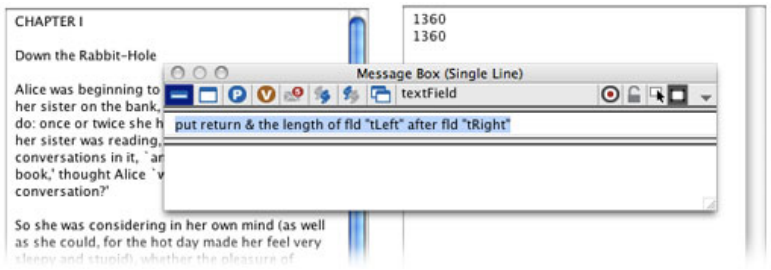
フィールド「tLeft」に何文字あるかフィールド「tRight」に入れなさい。
短縮形の言葉を使っても書けます。number = num, field = fld
put the num of chars of fld "tLeft" into fld "tRight"

もうひとつ「length（長さ）」と言う言葉でも同じ結果が得られます。右のフィールドの文字列の後に、行替えして（return キャラクターを入れて）その結果を書き出します。「field」を「fld」の短縮形で書いています。

```
put return & the length of fld "tLeft" after fld "tRight"
```

行替えとフィールド「tLeft」の「the length（長さ）」をフィールド「tRight」の後に入れなさい。

図：「length」を使って、フィールドの最後の行に追加



次に何行のライン（the number of lines）が、フィールドにあるのか調べてみます。ラインの数を求める時はラインは複数形「lines」です。「field」を「fld」の短縮形で書いています。

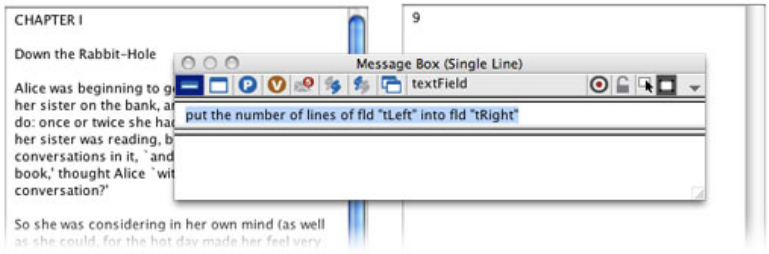
```
put the number of lines of fld "tLeft" into fld "tRight"
```

フィールド「tLeft」の行数をフィールド「tRight」に入れなさい。

「number」の短縮形を使うとこうなります。

```
put the num of lines of fld "tLeft" into fld "tRight"
```

図：ライン数を右のフィールドに書きます。

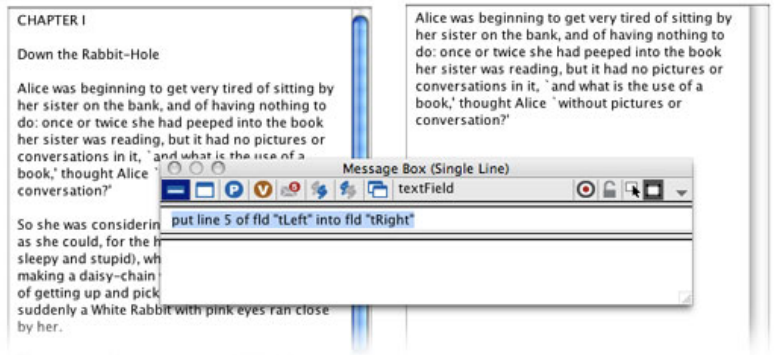


これから分かる事はアキの行も1行と数えられますから、「line 1」は「CHAPTER 1」ですが、「line 2」は空白で、「line 3」は「Down the Rabbit-Hole」になります。では「line 5」を右のフィールドに入れてみます。この場合のラインは、ラインの数ではないので単数形です。「field」を「fld」の短縮形で書いています。

```
put line 5 of fld "tLeft" into fld "tRight"
```

フィールド「tLeft」の5行目をフィールド「tRight」に入れなさい。

図：「line 5」を右のフィールドに入れます。



「characters」「length」「lines」と、もうひとつテキストをカウントする言葉に「words」があります。これも「number」と組み合わせて、下の例文のように使います。

```
put the number of words of fld "tLeft" into fld "tRight"
```

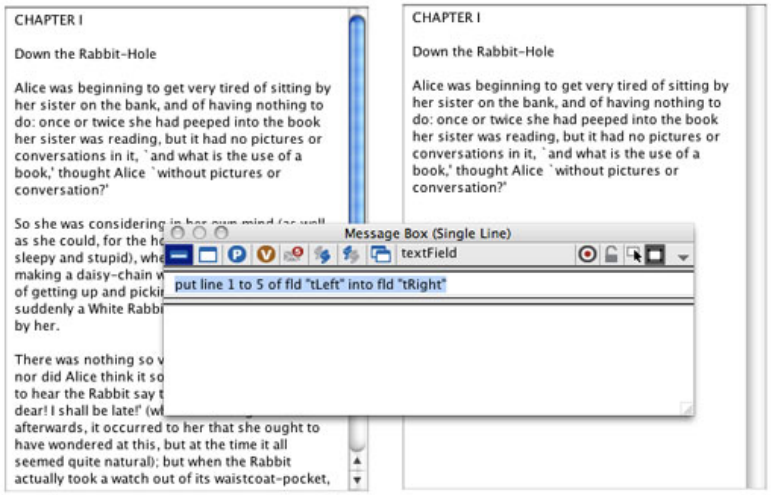
フィールド「tLeft」の単語数（the number of words）をフィールド「tRight」に入れなさい。

```
put word 3 of line 3 of fld "tLeft" into fld "tRight"
```
フィールド「tLeft」の「line 3」の「word 3」をフィールド「tRight」に入れなさい。
-- 結果は「Rabbit-Hole」が入ります。

フィールド「tLeft」のライン1からライン5までを、フィールド「tRight」に入れます。

```
put line 1 to 5 of fld "tLeft" into fld "tRight"
```
フィールド「tLeft」の「line 1 から 5 まで」をフィールド「tRight」に入れなさい。

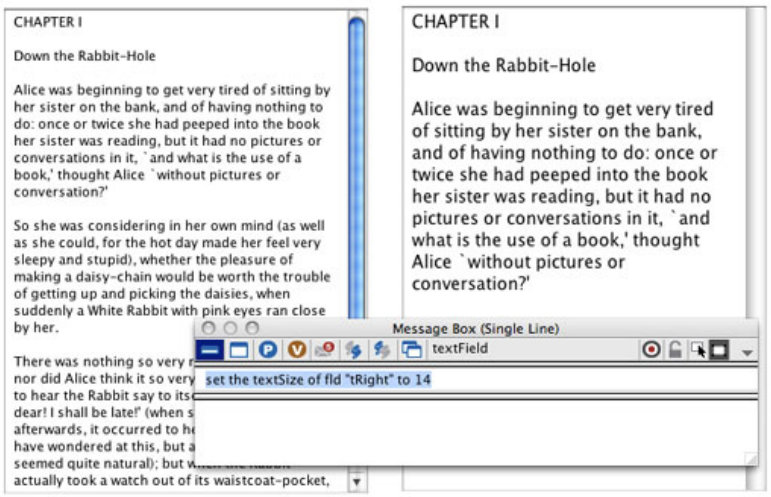
図：「line 1 から 5」を右のフィールドに入れます。



フィールド「tRight」のテキスト・サイズを「14」にします。
LiveCodeではフォント・サイズではなく「textSize　テキスト・サイズ」と言います。

```
set the textSize of fld "tRight" to 14
```
フィールド「tRight」のテキスト・サイズを「14」に設定しなさい。

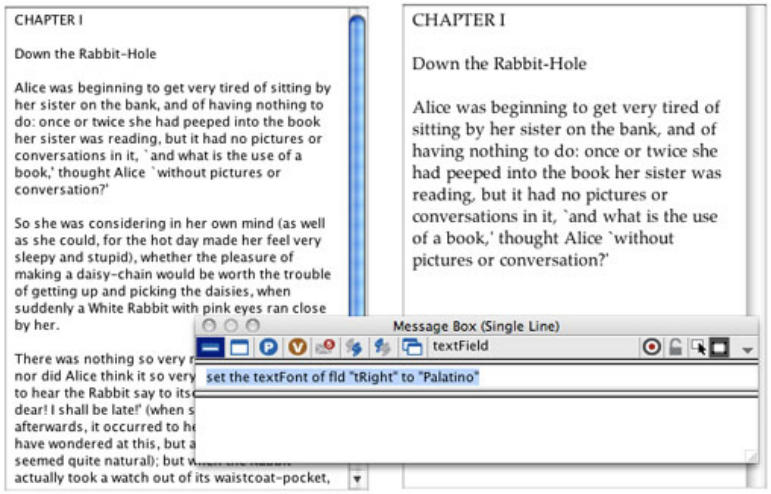
図：フィールド「tRight」のテキスト・サイズを「14」



フィールド「tRight」のテキスト・フォントを「Palatino」にします。
LiveCodeではフォントは「textFont　テキスト・フォント」と言います。

```
set the textFont of fld "tRight" to "Palatino"
```
フィールド「tRight」のテキスト・フォントを「Palatino」に設定しなさい。

図：フィールド「tRight」のテキスト・フォントを「Palatino」

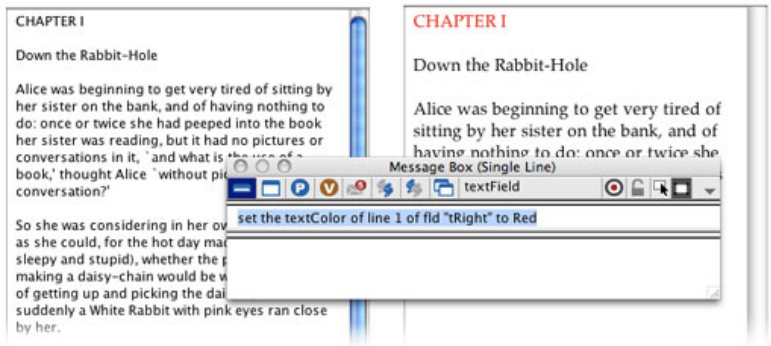


フィールド「tRight」の始めの行（line 1）のカラーを「Red　赤」にします。ここでは「textColor テキスト・カラー」と言う言葉を使いましたが、グフィックのカラー設定で使った「foregroundColor」と言う言葉でも、同じ意味で使う事ができます。またRGB値でのカラー設定もできます。

```
set the textColor of line 1 of fld "tRight" to Red
```

フィールド「tRight」のライン１のテキスト・カラーを「Red 赤」に設定しなさい。

図：フィールド「tRight」のライン１のテキスト・カラーを「Red」に

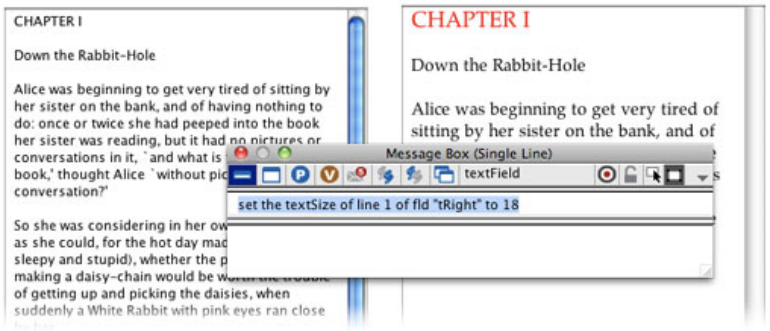


フィールド「tRight」全体のテキスト・サイズは14ポイントです。フィールド「tRight」の始めの行（line 1）のテキスト・サイズだけを18ポイントに変更します。

```
set the textSize of line 1 of fld "tRight" to 18
```

フィールド「tRight」のライン１のテキスト・サイズを18ポイントに設定しなさい。

図：フィールド「tRight」のライン１のテキスト・サイズを18ポイントに

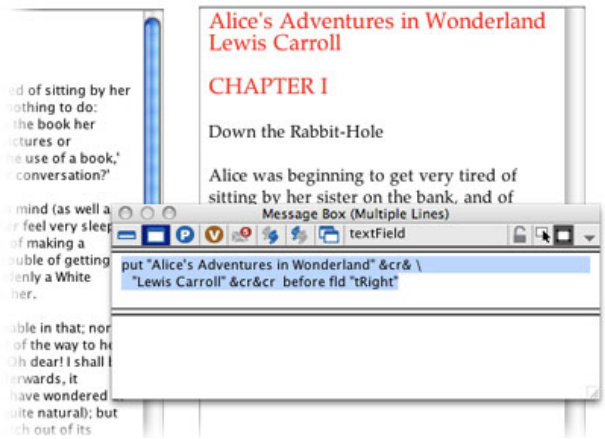


タイトルの「Alice's Adventures in Wonderland」と、行を変えて作者の名前「Lewis Carroll」を「before」を使って始めの行に挿入します。ここでは「return」と同義語の「CR（Carriage Returnの略）」を使っています。「Carriage Return」は、昔風なタイプライターが紙を支えながら行替えするメカニズムを表す用語でしたが、コンピュータ時代にも言葉だけが残って、プログラミングでは行替えキャラクターとして「CR」と使われる事が多いです。「Lewis Carroll」の後に1行のをアキを取るので「& cr & cr」と行替えを2度行います。もうひとつ、あまり長過ぎるスクリプトの行が読みにくくなったり、エディターの横に入らなくなったらしたら、「\　バックスラッシュ」を行を変える最後に入れて行替えします。フィールドの横巾にタイトルが入りきらないので、手動でスタックの横を少し伸ばしました。

```
put "Alice's Adventures in Wonderland" &cr& \
   "Lewis Carroll" &cr&cr  before fld "tRight"
```

「Alice's Adventures in Wonderland」と行替えと「\　バックスラッシュ」
「Lewis Carroll」と行替え2回をフィールド「tRight」の前に入れなさい

図：フィールド「tRight」にタイトルと作者名を挿入



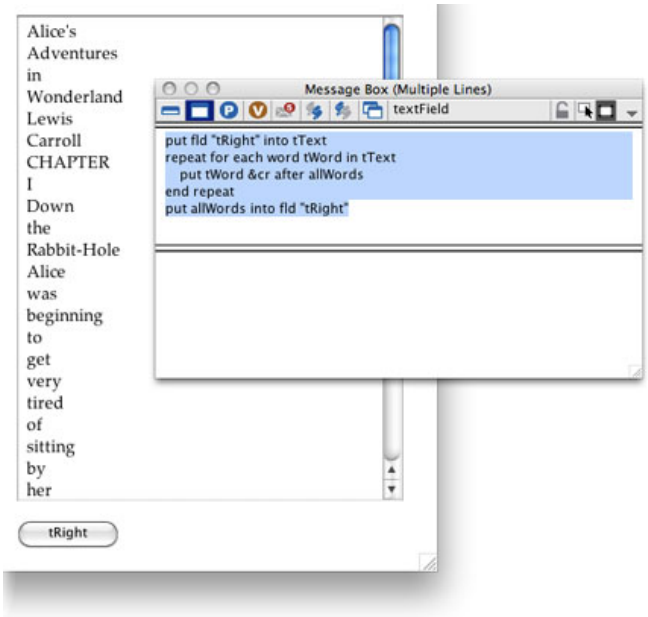
高速なリピートと、アルファベット順にソートする

今度は単語をバラバラにして、フィールドに並べてみます。リピートで単語（word)を1つづつ各行に置いて行く作業ですから、上で書いた方法で単語の総数を数えて、「repeat with i=1 to 単語の総数」と言う前章のループも使えますが、ここではより高速なループの処理をする「for each word」や「for each line」、「for each character」、「for each item」などの単位を区切りに使う方法をとります。マルチライン・メッセージ・ボックスから、フィールド「tRight」にある文章の全てを、単語別の行にするスクリプトを送ります。

```
put fld "tRight" into tText
repeat for each word tWord in tText
    put tWord &cr after allWords
end repeat
put allWords into fld "tRight"
```

フィールド「tRight」のテキストを「tText」に入れます。
「tText」の単語（word）それぞれ1つづつを「tWord」と言うコンテナーに入れてリピートします。
「tWord」と行替え文字の「cr」を、「allWords」と言うコンテナーの後ろに入れて行きます。
全ての単語のリピートが終わったらリピートを終わります。
コンテナー「allWords」をフィールド「tRight」に入れます。

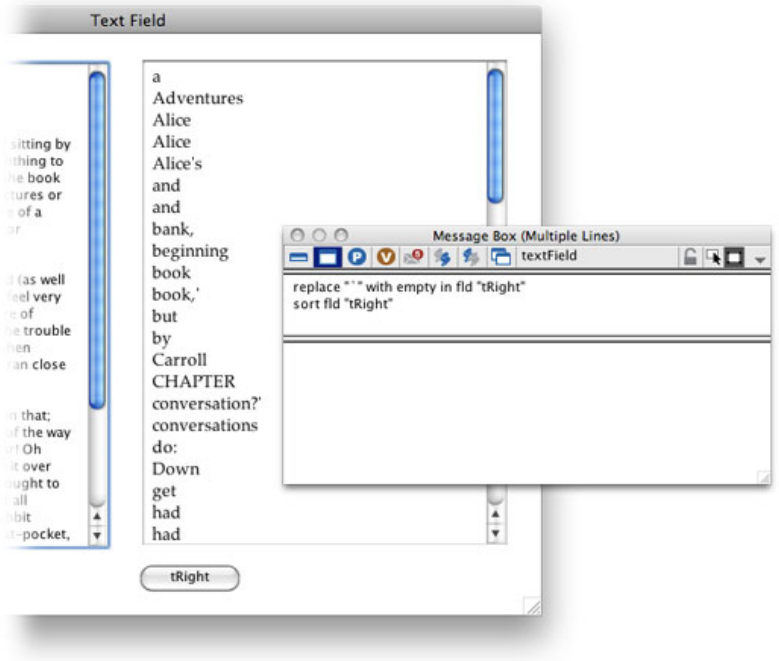
図：リピートで全ての単語を書き出す



これで全ての単語がフィールドに書き出されました。次にアルファベット順に単語を並べ替えたいのですが、下の方に2つ「`and」と「`without」と言う始めに記号の付いたのがありますから、その記号部分を取り除いてアルファベットだけで始まる単語にしてから、並べ替え（ソート　sort）をします。

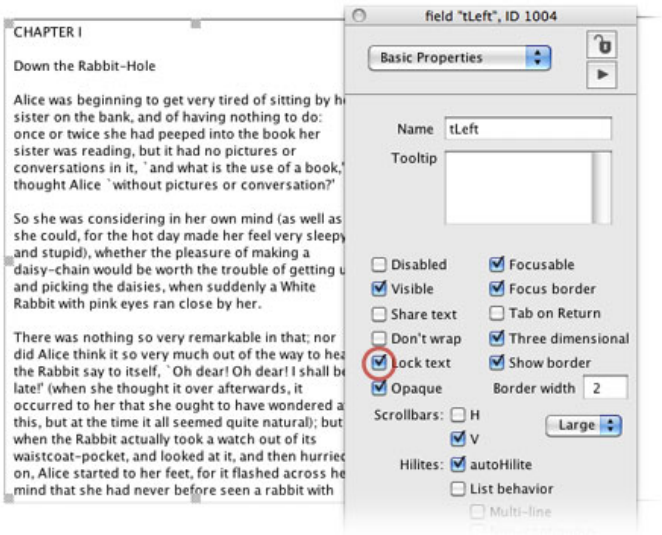
```
replace "`" with empty in fld "tRight"
```

図：単語をソート



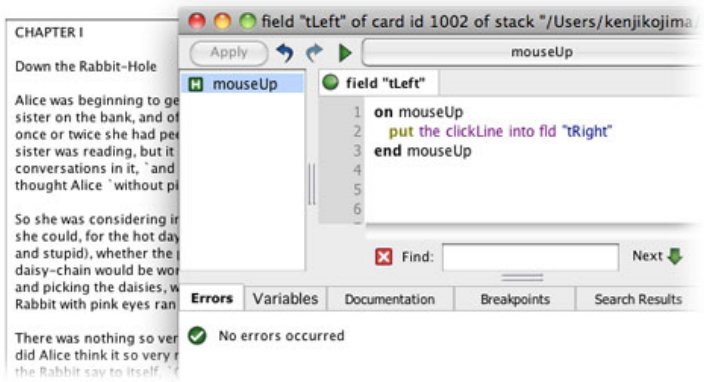
フィールドのエディターにスクリプトを書く

「算数クイズ」の答えのフィールドのスクリプト・エディターに、「returnInField」と言うハンドラーのスクリプトを書きました。今度はフィールドにブラウズ・ツールのマウスが入れられないようにして、キャラクターや文字をクリックした時のスクリプトを書きます。左のフィールド「tLeft」の「Lock text」にチェックを入れてください(下図赤丸）。



ブラウズ・ツールを持ったままでスクリプト・エディターを開けるのは、Mac OSは「オプション＋コマンド・キー」、Windowsは「コントロール＋オルト・キー」で、オブジェクトをクリックします。始めに「on mouseUp」を書き込んで「リターン・キー」を叩くと、1行分アキを取って「end mouseUp」が自動で書かれます。「clickLine」を試してみましょう。

```
on mouseUp
   put the clickLine into fld "tRight"
end mouseUp
```



上のように書き込んだら、エディター左上の「Apply」をクリックしてから、フィールドをブラウズ・ツールでいろいろな箇所をクリックしてみます。始めの行の「CHAPTER I」をクリックすると「line 1 of field 1」が出てきます。アキの行でも「line 4 of field 1」のように、全てのフィールドにある行の位置が返されます。

ここから実際の行の文字列を書き出してみましょう。「the clickLine」の前に「the text of 」を入れて

```
on mouseUp
   put the text of clickLine into fld "tRight"
end mouseUp
```

とフィールド内を書き換えると、今度はクリックしたラインの文字列が書き出されます。次に「clickChunk」と言う言葉を使ってみます。「chunk」はひとかたまりと言う意味で、ひとかたまりのキャラクター・ナンバーを返します。ほとんどの場合このひとかたまりは、単語（word）と解釈して良いでしょう。「char 44 to 52 of field 1」などのキャラクター・ナンバーを、フィールド「tRight」に書き込みます。「char」は「character」の短縮形です。

```
put the clickChunk into fld "tRight"
```

```
put the text of clickChunk into fld "tRight"
```

こちらも「the text of clickChunk」にすると、実際の文字列が返されますが、上で「word　ワード」を右のフィールドに書き込んだ時は「Rabbit-Hole」をひとつの単語としていましたが、「clickChunk　クリック・チャンク」では「Rabbit」「-」「Hole」はそれぞれ独立した「chunk（ひとかたまりの文字列）」として捉えられます。

もうひとつキャラクターについて書いておきます。フィールド内のキャラクターのナンバーは「clickCharChunk」で得られます。使われている「char」は「character」の意味で、スクリプト中でも「char」の短縮形で使われます。

```
put the clickCharChunk into fld "tRight"
```

例えば、1行目の「CHAPTER I」の最後の「I」は「char 9 to 9 of field 1」が返されます。その前にあるスペースは「char 8 to 8 of field 1」で、3行目の始め「D」は「char 12 to 12 of field

1」です。つまり「space　スペース」も、「return　リターン（CR）」も1キャラクターと数えられます。実際のキャラクターそのものは「clickChar」で得る事ができます。

```
put the clickChar into fld "tRight"
```

プログラミングをして行くと、幾つかの文字コードを知っておく必要が出てきます。ASCII（アスキー）は、ヨーロッパ言語で使われるアルファベット、数字を中心とするキャラクターに数字を割り当てた文字コードです。キャラクターからASCII（アスキー）のナンバーを知るには「charToNum」と言うファンクションを使って小文字の「a」でしたら「charToNum("a")」のように使います。「the clickChar」を使ってASCII（アスキー）のナンバーを知るには

```
put charToNum(the clickChar) into fld "tRight"
```

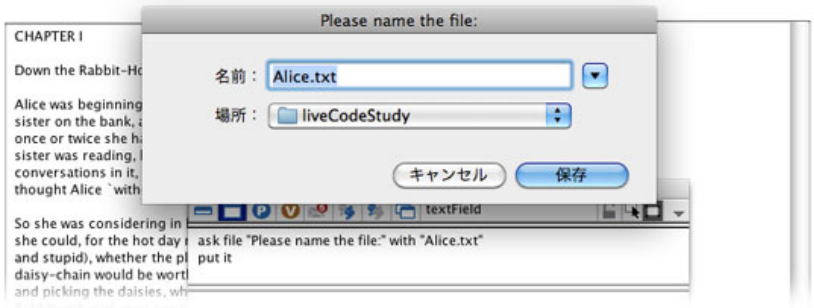
反対にナンバーからキャラクターを知るのは「numToChar(97)」のように使います。これで小文字の「a」が返されます。例えばスクリプトを書いていて、前後の事情でキャラクターが直接書き込めない時などに、「numToChar(10)」のような形をとって使う事があります。今の処はこういう文字コードがあって、互いに変換できると言う事を知っているだけで良いでしょう。

フィールドの英語テキストを保存する

マルチライン・メッセージ・ボックスから

```
ask file "Please name the file:" with "Alice.txt"
put it
```

を送ると、ファイルを保存するダイアログが出ます。



「キャンセル」をクリックすると、「it」はないも返しません。つまり「empty 空」を返します。もう一度メッセージ・ボックスから同じスクリプトを送って、今度はどこに保存するか「場所：」のフォルダーを選んで、「名前：」のファイル名「Alice.txt」はデフォルトのまま、ダイアログ右下「保存」を叩くと、メッセージ・ボックスに（私の場合は）「/Users/kenjikojima/Desktop/liveCodeStudy/Alice.txt」が返されました。「保存」でスクリプトの「it」から返される最後がファイル名になっている文字列を、私たちはファイル・パスと呼んでいて、これでどこにファイルを保存するかの指示をコンピュータに与えます。Windowsの場合は、通常こういうパスは日本語OSでは「￥　円記号」（英語OSでは「\　バックスラッシュ」）を使いますが、LiveCodeではMac OS、Windows共に普通の「/　スラッシュ」を使います。この場合のテキスト・ファイルの名前には必ず「.txt」が必要です。

メッセージ・ボックスから送ったスクリプトの説明をします。「ask file ""」と言うのが、ファイルを保存する基本の形で、クオートの中に入れる文（ストリング）は、ダイアログ・ウインドウのタイトルになります。英文であればその時々の状況でどんな文にでもできます。その後の「with」に続くクオートの中は（例では「Alice.txt」）は、デフォルトのファイル名にします。「with」から後は、スクリプトに書いても、書かなくても良いです。現バージョンのLiveCodeは、ダイアログ・ウインドウのタイトルも、ファイル名も日本語にはできません。

左下にあるボタン「tLeft」のレイベル（Label）を、インスペクターで「保存」にします。テキストをファイルにしてローカル・ディスクに保存するには、「url ("file:" & ファイルパス)」を使います。「url 」は、ローカル・ディスクへのテキスト・ファイルの保存だけでなく、インターネットでファイルを送ったり、受けたりする時にも使われます。それでは、メッセージ・ボックスでテストした結果を考えながら、スクリプトを書き込みます。「--」の後の日本語は説明（コメント）です、スクリプト・エディターには書き込めません。

```
ask file "Please name the file:" with "Alice.txt"
if it is empty then
  exit to top  -- 全ての実行を中止する
else
  -- もしファイル名に「.txt」が付いていなかったら、it の後ろに「.txt」を付ける
  if character -4 to -1 of it is not ".txt" then put ".txt" after it
  put field "tLeft" into url ("file:" & it)
end if
```

```
ask file "Please name the file:" with "Alice.txt"
if it is not empty then
   if character -4 to -1 of it is not ".txt" then put ".txt" after it
   put field "tLeft" into url ("file:" & it)
end if
-- 「exit to top」の考え方を理解する為に「if it is empty then」にしています。
```

ダイアログ・ウインドウの「キャンセル　Cancel」がクリックされて「it」が空（empty）で返されたら、全ての実行を中止してトップに出ます。そうでなかったら（else）ファイル・パス（it）の最後にあるファイル名に「.txt」があるかどうかをチェックして、もしなかったら「it」で得たファイル・パスの最後に「.txt」を付けます。文字列の先頭のキャラクターは「character 1」と認識されて、次々にナンバーが振り当てられます。もうひとつ文字列のキャラクター認識は最後から「character -1」と番号が割り振られ、前に進むにしたがってマイナスのナンバーが増えて行く、システムになっています。従って最後に付いている「.txt」は「character -4 (.)」から「character -1 (t)」までの4文字ですから「character -4 to -1 of it」となります。これまで「if文」は「then」で行替えして、最後は「end if」で終わる形式を使ってきましたが、このように1行で書ける「if文」は、最後の「end if」を取り除く事ができます。

「url ("file:" & ファイル・パス)」はプレインな英語テキストの保存に使います。それ以外のイメージや日本語など、バイナリー・データ（プレイン・テキスト以外のデータ）の保存では「url ("binfile:" & ファイル・パス)」を使います。詳しくは後の章で。

ジオメトリーをスクリプトで書く

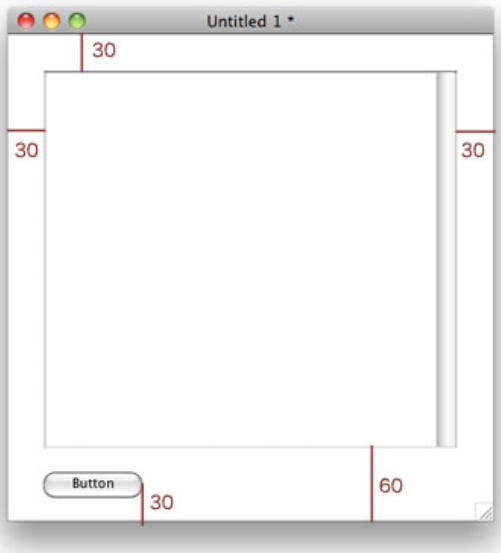
フィールドの章の最後に、スタックのリサイズに沿ってフィールドのサイズを変えたり、ボタンのロケーションを変えるスクリプトを書いておきます。ジオメトリー・マネイジーによらない、リサイズです。新しいメイン・スタックを作って、フィールドをひとつ、ボタンをひとつ作ったら、「resizeStack」ハンドラーをスタックのスクリプト・エディターに書き込みます。スタックのスクリプト・エディターを開けるのは「Objectメニュー > Stack Script」を選ぶか、アイコン・メニューの「Code」をクリックします。下の緑の文字すべてをスタック・エディターに書き込みます。

```
on resizeStack pWidth, pHeight
   set the location of button 1 to \
      (30 + the width of button 1 div 2), pHeight -30
   set the rect of field 1 to 30, 30, pWidth -30, pHeight -60
end resizeStack
```

```
on ハンドラー「resizeStack」  pWidth(パラメータ横), pHeight(パラメータ縦)
  （スタックの縦横の変化によって）
  ボタン１のロケーションは（30 + ボタンの横の1/2), pHeight - 30に設定
  フィールド１のレクト（rect  矩形）を \
     30,30,pWidth（スタックの横）-30, pHeight（スタックの縦）-60に設定
end ハンドラー「resizeStack」
```

「on resizeStack pWidth, pHeight」の「pWidth」はスタックの横のピクセル数、「pHeight」はスタックの縦のピクセル数をパラメータに取ります。この「pWidth, pHeight」はユーザーがスタック・サイズを変えるごとにいつも変化します。図で赤で「30」とある場所は、常に30ピクセルのアキを保ちます。フィールドの底辺は常に60ピクセルです。ボタンのロケーションのX値は、フィールドの左横のアキ（30ピクセル）に、ボタンの横巾の半分を足したピクセル数です。

図：スクリプトを使ったスタックのリサイズ、



注：リサイズスタック（resizeStack）・ハンドラーを使うと、ジオメトリー・マネジャーの設定は無効になります。

Tips

- 1行が長過ぎて読みにくくなったりしたスクリプトは、行を変えたい最後に「\　バックスラッシュ」を入れて行を変えます。

- 「word　ワード」や「line　ライン」や「item アイテム（comma　カンマ）」などで区切れるデータのループは、「repeat for each 区切りの語」を使うと高速な処理が行えます。

- 行替えのキャラクター「return」はしばしば、「Carriage Return」の短縮形「cr」が使われます。

- ストリング（文字列）の先頭のキャラクターには「character 1（char 1）」の数字が割り当てられ、次々に番号が増えて行くシステムが取られています。先頭から10番目のキャラクターは「char 10」。同じようにストリングの最後のキャラクターには「char -1」が割り振られて、前に進むに従ってマイナスの番号が増えるシステムも、同時に取られています。最後のキャラクターから数えて5番目は「char -5」。

この章で新しく出て来た言葉

vScrollbar
プロパティ（property）縦のスクロールがあるかを返す
```
put the vScrollbar of field 1
```

replace
コマンド（command）コンテナーにあるテキストを入れ替える
```
replace return with return & return in it
```

characters
キーワード（keyword）
ソート・コマンドやナンバー・ファンクションにある複数の文字を示す　短縮形：char
```
put the number of characters of field "tLeft" into field "tRight"
```

length
ファンクション（function）ストリング中にある文字数
```
put the length of fld "tLeft"
```

number
ファンクション（function）
オブジェクトの数やストリング中の文字数　短縮形：num
```
the number of words of fld "tLeft"
```

textSize
プロパティ（property）テキストを表示するポイント・サイズ
```
set the textSize of fld "tRight" to 14
```

textColor
プロパティ（property）テキストを表示するカラー　同義語：foregroundColor

```
set the textColor of line 1 of fld "tRight" to Red
```

before
キーワード（keyword） putコマンドと使ってフィールドの特定な前の箇所を指し示す
```
put "Alice's Adventures in Wonderland" &cr before fld "tRight"
```

each
キーワード（keyword） repeatループの実行でそれぞれのラインやワード等の区切りを示す
```
repeat for each word tWord in tText
```

after
キーワード（keyword） putコマンドと使ってフィールドやストリングの特定な後ろの箇所を指し示す
```
put tWord &cr after allWords
```

clickLine
ファンクション（function） クリックされたフィールドのライン・ナンバーを返す
```
put the clickLine
```

clickChunk
ファンクション（function） クリックされたフィールドの語（word）か文字のグループを返す
```
put the clickChunk
```

clickCharChunk
ファンクション（function） クリックされたフィールドの文字（character）のポジションを返す
```
put the clickCharChunk
```

charToNum
ファンクション（function） キャラクターのASCII ナンバーを返す
```
put charToNum("A")
```

numToChar
ファンクション（function） ASCII ナンバーのキャラクターを返す
```
put numToChar(65)
```

ask file
コマンド（command） ファイル保存のダイアログを表示する
```
ask file "Please name the file:" with "Alice.txt"
```

is not
オペレータ（operator） ふたつの値を比べて等しくなければ真　同義語：<>
```
if character -4 to -1 of it is not ".txt" then put ".txt" after it
```

url
キーワード（keyword） コンテナーにローカルかインターネット・ファイルが含まれているのを意味する
```
put field "tLeft" into url ("file:" & it)
```

resizeStack
メッセージ（message） スタックのウインドウ・サイズが変わったかコードに送る
```
on resizeStack pWidth, pHeight
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

11: ２種類のイメージ

この章の概略

- *テキスト・データを含んでいるイメージ・オブジェクトと、リンクしたイメージ・オブジェクトの違いと扱い方。*
- *デフォルト・フォルダー（DefaultFolder）とスペシャル・フォルダー・パス（specialFolderPath）。*
- *幾つも並んでいる項目を区切るキャラクター「itemDelimiter　アイテム・デリミター」の変更。*

この章で使われる英単語

- template [名詞：テンプレート, 型]
- fill [動詞：いっぱいに満たす]
- format [動詞：書式を整える]
- option [名詞, 動詞：選択（する）]
- delete [動詞：削除（する）]
- default [名詞：（第一義的には）不払い,（コンピュータでは）初期設定]
- reference [動詞：参照する]
- delimit [動詞：範囲を定める]

注意：このチュートリアルは、LiveCodeでアプリケーションを開発して行く上で、絡み合っている事柄をステップ・バイ・ステップで、用語を積み上げて解きほぐしているので、興味あるタイトルの章だけを抜粋して拾い読みして行くと、大事な事柄を飛ばしているかもしれません。

写真はラスター・グラフィック画像（またはビットマップ画像）と言われる、沢山のピクセル・ドットの集まりで表現します。最近は時々しか見ないですが、以前はお絵描き（ペイント）プログラムと言うのポピュラーでした。お絵描きでは線や面が単色でベタな感じで、写真は1つ1つのピクセルを細かな色の違いで表現します。どちらも基本的には同じものです。LiveCodeではイメージ（image）と言うオブジェクトをカード上に作って、その中で画像を見せます。イメージ・オブジェクトの外には画像は表現できません。LiveCodeでは例外的に、テキスト・フィールドの中のキャラクターにイメージ表示をする、「imageSource」と言う表現もできますが、ここでは扱いません。

お絵描き（ペイント）のイメージ・オブジェクト

始めにペイント・イメージのサンプルです。スクリプトでカード上にイメージ・オブジェクトを作って、その中にブルーでバツ印を大きく描きます。ニューメイン・スタック（New Mainstack）をFileメニューから作ってください。メッセージ・ボックスから下のスクリプトを送ります。マルチライン・メッセージから1度に送っても良いですし、１行づつスクリプトを確認しながら送っても良いです。１行づつ送ると、カーソルが途中でブラッシュの丸い形に変わりますから、そのつもりで。

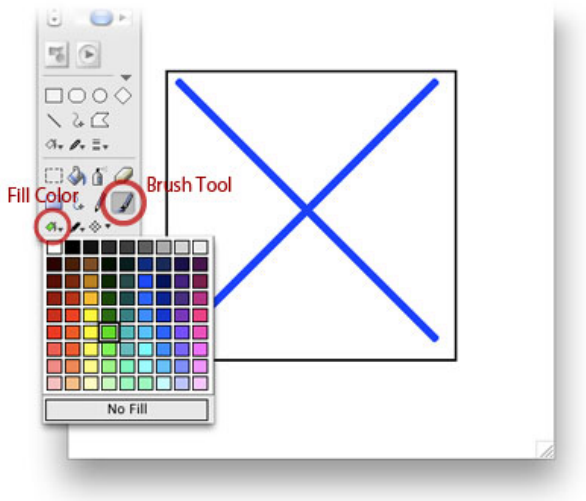
```
set the rect of the templateImage to 80,80,320,320
create image
set the showborder of the last image to true
set the threeD of the last image to false
choose brush tool
set the brush to 8
set the brushColor to blue
```

```
set the dragSpeed to 120
drag from 90,90 to 300,300
drag from 90,300 to 300,90
choose browse tool
```

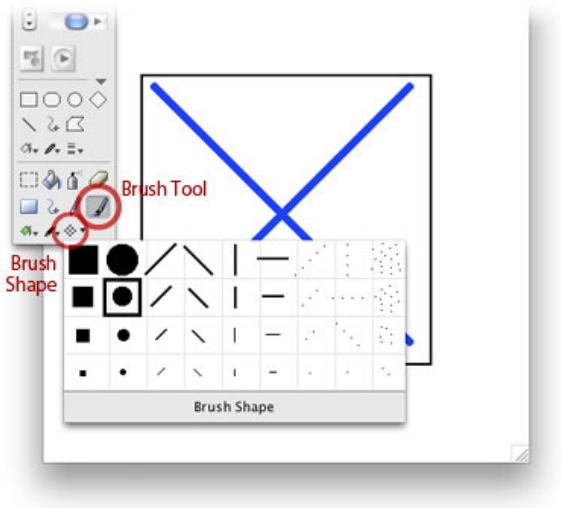
デフォルト・イメージ・オブジェクトのレクト（rect)を80,80,320,320に設定。
イメージ・オブジェクトを作りなさい。
最後に作ったイメージを、ボーダーが見える設定にします。
最後に作ったイメージの、3Dボーダーの設定を負にします。
ブラッシュ・ツールを取りなさい。
ブラッシュ（のサイズ）を8に設定しなさい。
ブラッシュのカラーをブルーに設定しなさい。
（ブラッシュの）ドラッグするスピードを120に設定しなさい。
90,90から300,300に（ブラッシュを）ドラッグしなさい。
90,300から300,90に（ブラッシュを）ドラッグしなさい。
ブラウズ・ツールを取りなさい。

1行1行の積み重ねは、簡単な言葉でできています。「the templateImage」は、デフォルトで作られるイメージ・オブジェクトのプロパティを設定します。3行目4行目で「the last image」としているのは、一番始めに作られるイメージ・オブジェクトは「image 1」ですが、それ以降のイメージ・ナンバーは不確かなので、どんな場合でも使える「最後のイメージ　the last image」と言う表現にします。

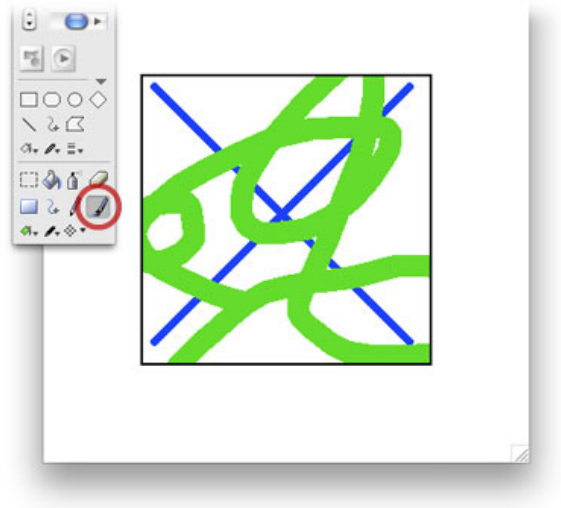
スクリプトでイメージ・オブジェクトにペイントしたので、今度はツール・パレットのペイント・ツールで、今作ったイメージ・オブジェクトの上に描き足してみます。まずブラッシュ・ツール（Brush Tool）をクリックで選んで、フィル・カラー（Fill Color）をクリックしたまま（マウスダウン）にすると、カラー・チャートで出ますからグリーンを選びました。フィル・カラーはブラッシュに色を満たす、というような意味合いでしょうか。



フィル・カラーで色を選んだら、ブラッシュ・シェイプ（Brush Shape）からのブラッシュ形と大きさを選びます。少し大きめの丸を選びました。



フリーハンドで適当に、イメージ・オブジェクトの中でマウスを動かしてみます。オブジェクトのある範囲の中でだけ、ブラッシュで描く事ができます。



イメージ・オブジェクトの中には、ブルーのバツ印とフリーハンドで描いたグリーンの線のデータが書き込まれていますから、それをそっくりそのまま他のイメージ・オブジェクトに移してみます。スタックの横巾を伸ばして、始めに作ったイメージ1（image 1）の右に、ツール・パレットからイメージ2（image 2）を、ドラッグ＆ドロップして、メッセージ・ボックスから、下のスクリプトを送るとまったく同じイメージ（ペイント）がイメージ2に現れます。

```
put the text of image 1 into image 2
```

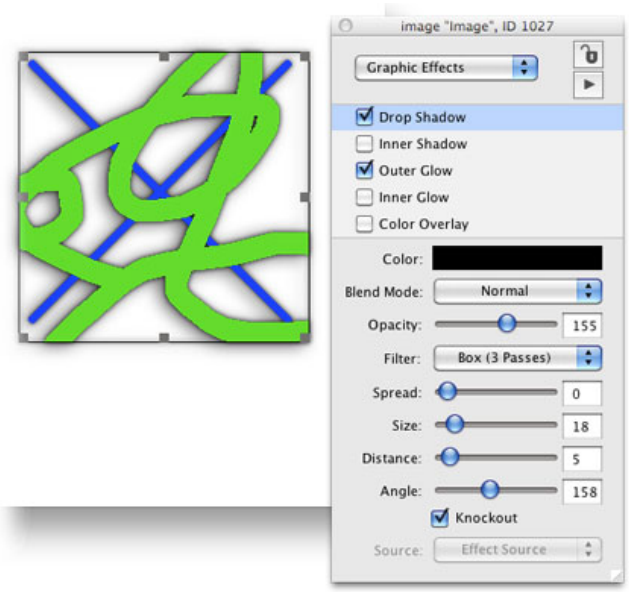
イメージ2（image 2）に入れたデータをメッセージ・ボックスで見てみます（下図）。

```
put the text of image 1
```

「the text」を外して、「put image 1」としても同じ結果が得られます。

何が何だかわかりませんが、とにかくこういうデータがイメージ・オブジェクトに書き込まれていると言うのがわかります。

グラフィック・イフェクト「Graphic Effects」



見た目には白地に色で描いたように見えますが、デフォルトのイメージ・オブジェクトは透明なバックグラウンドです。透明のオブジェクトがあるので、ついでにグラフィック・イフェクト（Graphic Effects）を試してみましょう。イメージ2をセレクトしたら、インスペクタのグラフィック・イフェクト（Graphic Effects）を開いて（上図）、適当に効果をいろいろ付けたり外したりして遊んでください。透明のバックグラウンドに描いているので、イメージ・オブジェクトの内部の透明部分にも効果が現れます。グラフィック・イフェクト（Graphic Effects）は、イメージ・オブジェクトだけでなく他のコントロールにも適用できます。ここではスクリプトでのグラフィック・イフェクトの記述は省きます。写真でこのような透明な効果を作り出すには、透明のあるPINGまたはGIFFイメージをインポートします。イメージのメニューからのインポートは、この章の下で書きます。

インターネットの写真をリンク

カード上にあるふたつのイメージ・オブジェクトの1つ「イメージ２」を削除します。セレクトしてデリート・キーで削除するか、下のスクリプトをメッセージ・ボックスから送ります。もし始めのスクリプトを何度も試していると、イメージ・オブジェクトが重なって沢山作られているかもしれません。何度も下のスクリプトを送って「no such object　そういうオブジェクトはありません」のエラーが出たら、イメージ1だけしか残っていないと言うことになります。

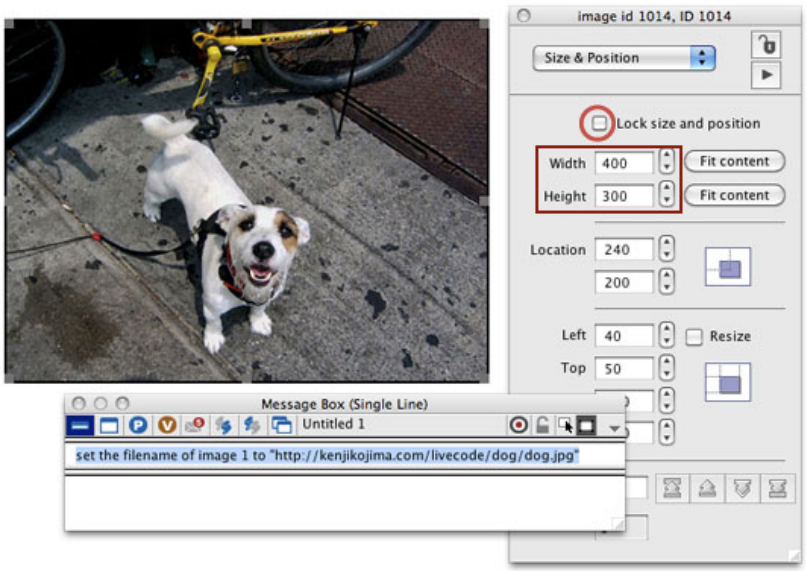
```
delete image 2
```

インターネット上にある写真 http://kenjikojima.com/livecode/dog/dog1.jpg を、下のスクリプトをメッセージ・ボックスから送って、イメージ・オブジェクト（image 1）にリンクします。

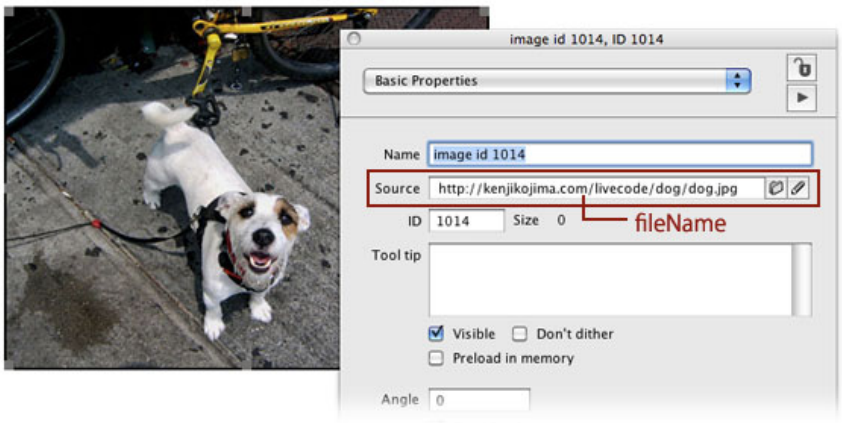
```
set the fileName of image 1 to "http://kenjikojima.com/livecode/dog/dog.jpg"
```

イメージ１のファイル・ネーム（fileName）を「http:...../dog.jpg」に設定しなさい。

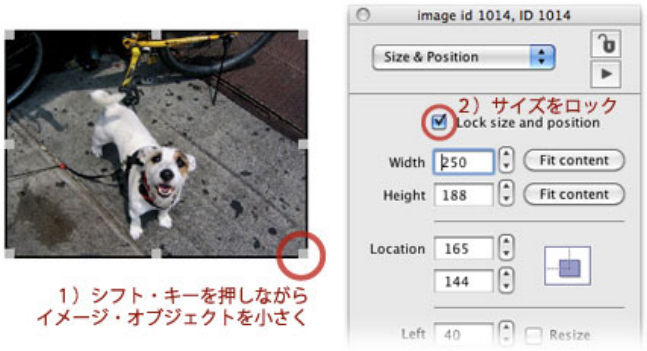
始めお絵描き（ペイント）のデフォルト・イメージの大きさ（rect　レクト)を80,80,320,320に設定していたので、サイズは240x240ピクセルでしたが、リンクしたイメージはその写真のオリジナル・サイズの400x300に変わります。サイズとポジションはロックされていません（下図：赤丸）。



ファイルをリンクする時に使った「fileName　ファイルネーム」は、インスペクタの「Basic Properties」に表示されます（下図：Source）。



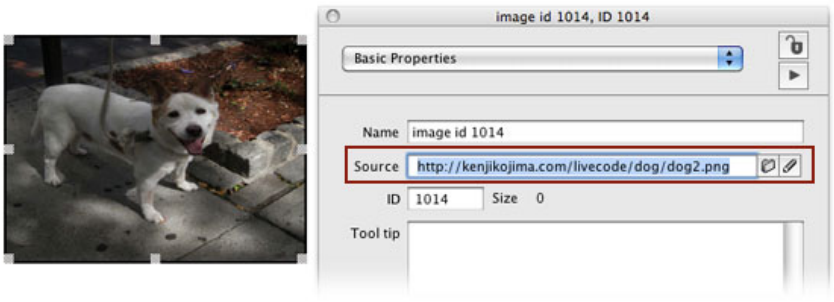
イメージのサイズを変えて、さらにそれをロックしてから、別なイメージをリンクします。シフト・キーを押しながら、写真の縦横比率が変わらないようにイメージの右下を縮小させて、インスペクタでロックします（下図）。



次に別なイメージをインスペクタからリンクしてみます。
先程はJPEGイメージでしたが、今度はPINGイメージ「dog2.png」です。

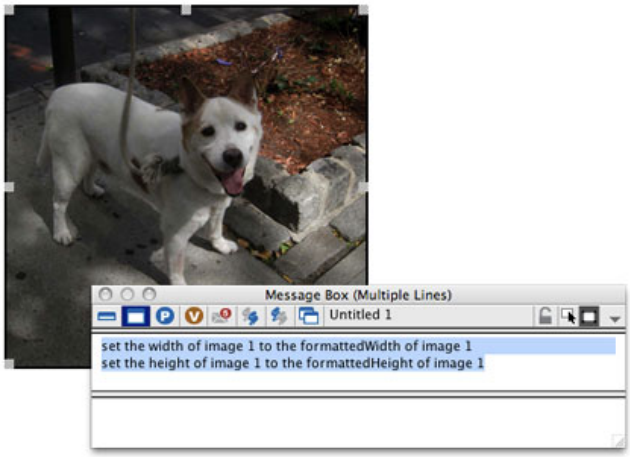
http://kenjikojima.com/livecode/dog/dog2.png

を、インスペクタの「Source」に上記を入れてリターン・キーを叩きます（下図：インスペクタからリンクする）。



たぶん気が付いた方もいるかと思います。ちょっとイメージが不自然です。実はイメージをロックしたまま、リンクをしたので画像がゆがんでいます。「formattedWidth」と「formattedHeight」を使ってオリジナルの画像サイズにします。メッセージ・ボックスから

```
set the width of image 1 to the formattedWidth of image 1
set the height of image 1 to the formattedHeight of image 1
```



「formatted　フォーマッテド」は書式を整えると言う意味で、「formattedWidth」と「formattedHeight」は、 オリジナル写真の左右・天地のピクセル数を返すプロパティです。

```
put the formattedWidth of image 1
```
イメージ１のフォーマッテド・ウイズ（オリジナル・ピクセル横）を返しなさい。

```
put the formattedHeight of image 1
```
イメージ１のフォーマッテド・ハイト（オリジナル・ピクセル縦）を返しなさい。

これでどちらも300を返します。リンクした写真は正方形にトリミングされたイメージです。イメージ・オブジェクトはロックしておくと自分の望む大きさにできるのですが、写真のプロポーションが崩れて、おかしなイメージになってしまいます。ですからリンクした直後に「formattedWidth」「formattedHeight」でオリジナルの大きさを知って、そのプロポーションで望むサイズにする方法を取ります。ロックされたイメージは、スクリプトで指示を与えるとサイズやポジションが変えられます。

下のスクリプトは、イメージの天地を180にして、オリジナルのプロポーションを維持します。オリジナル横 x 天地180 ÷ オリジナル縦 = 同じ比率の横巾　と言う計算です。

```
set the fileName of image 1 to \
    "http://kenjikojima.com/livecode/dog/dog2.png"
put the formattedWidth of image 1 into formattedW
put the formattedHeight of image 1 into formattedH
put formattedH * 180 div formattedW into tWidth
set the width of image 1 to tWidth
set the height of image 1 to 180
```

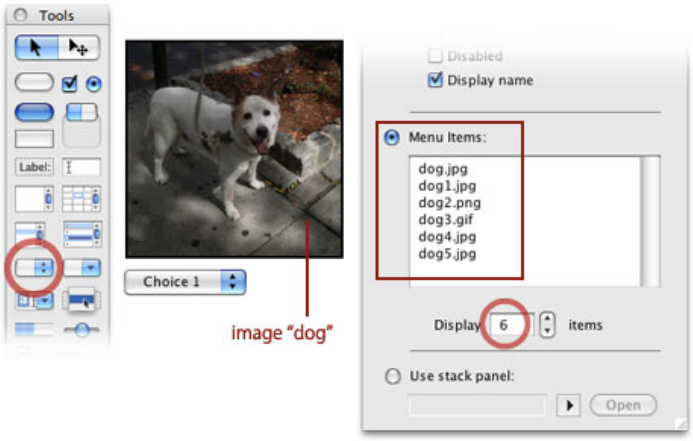
**イメージ1のファイルネーム（fileName）を **
**　　"http://kenjikojima.com/livecode/dog/dog2.png"に設定しなさい。**
イメージ1のオリジナル横巾をformattedWに入れなさい。
イメージ1のオリジナル天地をformattedHに入れなさい。
プロポーションを変えずに、イメージ1の天地を180にした時の横巾を、tWidthに入れなさい。
イメージ1の横巾をtWidthに設定しなさい。
イメージ1の天地を180に設定しなさい。

イメージのオリジナル・サイズは300x300の正方形でしたから、天地を180にすると、180の横巾になります。こういう場合は普通同じ比率のポイントを返すファンクションを作ります。上のスクリプトを、ファンクションを使ったボタンに入れるスクリプトにするとこうなります。イメージ 1 (image 1）の名前（name）を「dog」とします。

```
on mouseUp
   set the fileName of image "dog" to \
      "http://kenjikojima.com/livecode/dog/dog2.png"
   set the width of image "dog" to formattedW("dog",180)
   set the height of image "dog" to 180
end mouseUp

private function formattedW pImage, pHeight
   get the formattedWidth of image pImage
   return it * pHeight div the formattedHeight of image pImage
end formattedW
```

ファンクション「formattedW」で、天地180にした場合の同比率の横巾を返します。使っているパラメータ「pImage」は「イメージ・ネーム（またはイメージ・ナンバー）」、「pHeight」はイメージ・オブジェクトに設定する「天地のポイント数」です。

このスクリプトを使って、オプション・メニューからファイル名を選んで、イメージを表示するスクリプトにしてみましょう。ツール・パレットからボタン・メニューの「Option Menu　オプション・メニュー」を、イメージ「dog」の下にドラッグ＆ドロップしてください。

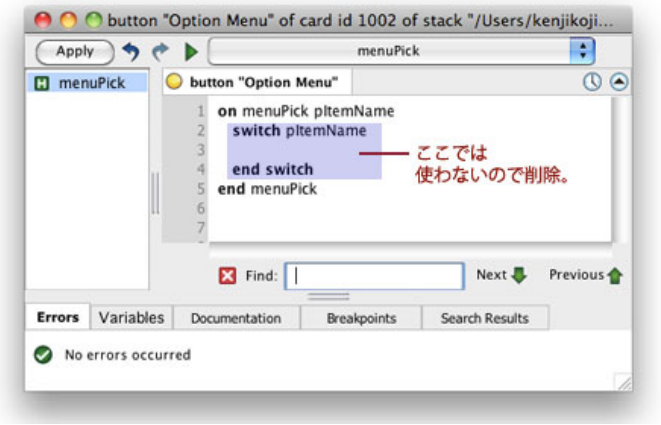


オプション・メニューのインスペクタを開いて、名前（Name）はそのままデフォルトの「Option Menu」で、インスペクタの下の方にあるメニュー・アイテムズ（Menu Items）に、この下にあるファイル名を入れて、Displayは「6」にします（上図：右赤丸）。Windowsではデフォルトの「5」にしておくと、6行目が隠れてスクロールしなくてはいけません。

```
dog.jpg
dog1.jpg
dog2.png
dog3.gif
dog4.jpg
```

dog5.jpg

オプション・ボタンをクリックして、メニューの項目がきちんと見えるようになったら、そのオプション・ボタン・メニューのスクリプト・エディターを開くと、下の図のように、「menuPick　メニューピック」メッセージ・ハンドラーの中に「switch　スイッチ」の文が書けるようになっています。普通メニューのような幾つかからの選択では「switch　スイッチ」を使う事が多いのですが、ここでは「menuPick　メニューピック」ハンドラーだけ残して、「switch」から「end switch」までは削除してください。「switch　スイッチ」は、別な章で話しましょう。



「menuPick　メニューピック」はユーザーがメニュー・アイテムから選んだ項目を、「on menuPick」の後にある、ここでは「pItemName」としているパラメータで返します。例えばユーザーが「dog4.jpg」を選んだとしたら、「pItemName」の中に「dog4.jpg」が入って、その後のスクリプトは「pItemName」を「dog4.jpg」として処理すると言う意味です。パラメータは「pWhich」でも、その他の言葉を使ってもかまいません。

上で使ったネット上の写真を、イメージ・オブジェクトにリンクするスクリプトの最後のファイル名を、ユーザーが選択するファイル名のパラメータ「pItemName」と差し替えます。ここではページ・レイアウトの関係で、バックスラッシュで行替えしています。ハンドラーの中の始めの文は１行で書いても問題ありません。

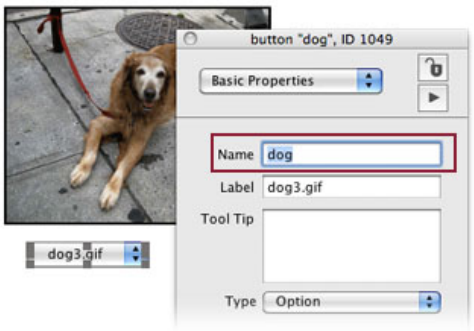
```
on menuPick pItemName
    set the fileName of image "dog" to \
         "http://kenjikojima.com/livecode/dog/" & pItemName
end menuPick
```

これで試してみると、写真の入れ替えはうまくいってるのは分かるでしょう。大きさの調整が必要ですね。次に作ったファンクション「formattedW」を加えます。

```
on menuPick pItemName
   set the fileName of image "dog" to \
         "http://kenjikojima.com/livecode/dog/" & pItemName
   set the width of image "dog" to formattedW("dog",180)
   set the height of image "dog" to 180
end menuPick
```

```
private function formattedW pImage, pHeight
   get the formattedWidth of image pImage
   return it * pHeight div the formattedHeight of image pImage
end formattedW
```

大きさは天地で揃うようになったので、もう少しポジションの調整を追加することにします。場合によったら、全て左揃えにしなくてはいけない事も出るかもしれません。ボタン・メニューの左を基準に整えて行きます。オプション・ボタン・メニューの名前をデフォルトの「Option Menu」から「dog」にします。



まずボタン「dog」の左上のロケーションを知るには、メッセージ・ボックスから

```
put the topLeft of button "dog"
```
ボタン「dog」のトップレフト（左上）を返しなさい。

簡単明瞭で、まったくそのままの意味です。X値とY値の2つのアイテムが、カンマで区切られて返されます。今度は、イメージ・オブジェクトの左下のロケーションを聞いてみましょう。

```
put the bottomLeft of image "dog"
```
イメージ「dog」のボトムレフト（左下）を返しなさい。

この図の例では、イメージ「dog」のボトムレフト（左下）はX値もY値も、ボタン「dog」より小さいです。ボタン「dog」と同じ位置で左揃えにするには、X値をボタン「dog」と同じにして、Y値は適当な間隔があく程度にボタン「dog」よりも小さくします（上にします）。だいたい「10」くらいのアキが良いでしょうか。

```
put the topLeft of button "dog" into tButtonTopLeft
set the bottomLeft of image "dog" to \
    item 1 of tButtonTopLeft, item 2 of tButtonTopLeft - 10
```

ボタン「dog」のトップレフト（左上）を「tButtonTopLeft」に入れなさい。
イメージ「dog」のボトムレフト（左下）を アイテム1の「tButtonTopLeft」と
アイテム2の「tButtonTopLeft」-10 にセットしなさい。

メッセージ・ボックスからスクリプトを送って、もしイメージ「dog」がトンデモナイ所に移動したら、「tButtonTopLeft」のアイテム（item 1 または item 2）が正しいか確認してください。イメージとボタンとの間隔は、「item 2 of tButtonTopLeft - 10」の数字の調整でします。

グラフィックの章でやったビジュアル・イフェクトを付ける事もできます。下のスクリプトがボタン「dog」に書き込む全てです。

```
on menuPick pItemName
   lock screen for visual effect
   set the fileName of image "dog" to \
         "http://kenjikojima.com/livecode/dog/" & pItemName
   set the width of image "dog" to formattedW("dog",180)
   set the height of image "dog" to 180
   put the topLeft of button "dog" into tButtonTopLeft
   set the bottomLeft of image "dog" to \
         item 1 of tButtonTopLeft, item 2 of tButtonTopLeft - 10
   unlock screen with visual effect "dissolve"
end menuPick

private function formattedW pImage, pHeight
   get the formattedWidth of image pImage
   return it * pHeight div the formattedHeight of image pImage
end formattedW
```

リンク・イメージの保存
デフォルト・フォルダー「DefaultFolder」
スペシャル・フォルダー・パス「SpecialFolderPath」

リンクしたイメージは、お絵描き（ペイント）したイメージ・オブジェクトのようには、オブジェクトの中身のデータを他のイメージ・オブジェクトに、そっくり移す事はできません。試みにこのスクリプトをメッセージ・ボックスから送ると、何もないempty（空）を返します。

```
put the text of image "dog"
```
イメージ「dog」の（中身の）テキストを返しなさい。

リンクしているイメージ・オブジェクトは、イメージの中継をしているだけですから、カード上にあるイメージをスナップ・ショットのように切り取って、新たなデータをエクスポートして保存します。リンクしたイメージをエクスポートしたファイルは、リンク元にあるオリジナルのサイズやファイル形式ではありません。今ここでは全てのイメージの天地を、180ピクセルにしていますから、その大きさで保存されます。保存できる形式は「JPEG, GIF, PNG, BMP」です。（オリジナルのイメージ・サイズ、形式で保存する方法はここではやりません）

```
export image "dog" to file "testImage.jpg" as JPEG
```
イメージ「dog」をJPEGフォーマットでファイル名「testImage.jpg」でエクスポートしなさい。

このスクリプトで、リンクしたイメージのファイルをJPEGで保存できます。しかし今のままの状態では、LiveCodeアプリケーションの置いてあるデフォルト・フォルダー（defaultFolder）に、保存されてしまいます。デフォルト・フォルダー（defaultFolder）は、下のスクリプトでパスを得られます。

```
put the defaultFolder
```
デフォルト・フォルダー（defaultFolder）のパスを書き出しなさい。

次に進む前に、少しデフォルト・フォルダー（defaultFolder）の説明をします。LiveCodeを開

くと、デフォルト・フォルダーはLiveCodeアプリのあるフォルダーになります。開発が完了して、LiveCodeプロジェクトを配布できるスタンドアロン・アプリにした場合は、そのスタンドアロンのある場所が、デフォルト・フォルダーになります。LiveCodeエンジンがあるフォルダーと言ってもよいでしょう。ですからイメージをエクスポートしたい場合、ユーザーが選んだフォルダーとか、デスクトップとかの適当な場所を、デフォルト・フォルダーに設定しないと、LiveCodeエンジンのあるフォルダーにファイルが入ってしまいます。

デフォルト・フォルダーは不便な事だけでなく、有用なこともあって、デフォルト・フォルダーに設定されていると、そのフォルダーの中のファイルの状態を知る事ができます。メッセージ・ボックスから

```
put the files
```

（デフォルト・フォルダーにある）ファイルを全て書き出しなさい。

と送ると、デフォルト・フォルダー（現在はLiveCodeアプリの置いてあるフォルダー）の中の、アプリケーションを除いたファイルのリストが返されます。もっと詳しい内容を知るスクリプトもありますが、ここでは省きます。ですから状況によってデフォルト・フォルダーを設定することで、テンポラリー・ファイルを保存したり削除したり、またフォルダーの中身を見たりする事ができます。

今はユーザーの指定なしに、デスクトップにイメージ・ファイルが保存できるようなボタンを作る事にします。ツール・パレットから角のある横長ボタン「Rectangle Button」をオプション・ボタンの右隣に置いてLabel（レイベル）は日本語で「デスクトップに写真を保存」とします。ただしMac OSのこのバージョン（6.0.1）で、ボタンへの直接日本語入力に問題があるようですから、うまく直接入力できない時はインスペクタのレイベル（Label）にコピペしてください（Windwosではまだ未確認）。ボタンは適当な横巾に伸ばしてください。

デフォルト・フォルダーを確認します。メッセージ・ボックスから

```
put the defaultFolder
```

デフォルト・フォルダー（defaultFolder）のパスを返しなさい。

デフォルト・フォルダーがどこにあるか、パスを返します。私は今Mac OSを使っていてアプリケーション・フォルダーの中の「LiveCode_CE」と言うフォルダーに入れているので、「/Applications/LiveCode_CE」が返ってきました。アプリケーション・フォルダーに直接LiveCodeアプリを入れている人は「/Applications」が返されます。

このパスをデスクトップに変えるのには、ファンクションの「specialFolderPath　スペシャ

ル・フォルダー・パス」を使います。ファンクションの「specialFolderPath」は、コンピュータを操作する上で、デスクトップやフォントやテンポ・ファイルの置き場所など、特定のフォルダーのある場所のパスを返します。MacかWindowsかによって、スペシャル・フォルダーに少し違いがあるので、今は共通に使える「Desktop　デスクトップ」を扱います。

```
put specialFolderPath("Desktop")
```
デスクトップ（Desktop）のパスを返しなさい。

これで私は「/Users/kenjikojima/Desktop」が返ってきました。当然ユーザーの違いで、返されるパスが違ってきます。デフォルト・フォルダー（defaultFolder）をスペシャル・フォルダー・パス（specialFolderPath）にセットすれば、エクスポートするイメージ・ファイルはデスクトップに現れます。

```
set the defaultFolder to specialFolderPath("Desktop")
```
デフォルト・フォルダー（defaultFolder）をデスクトップ（Desktop）に設定しなさい。

もう一つの問題は、ボタンをクリックするだけで、保存するファイル名を付けなくてはいけません。下のスクリプトの"testImage.jpg"の部分です。

```
export image "dog" to file "testImage.jpg" as JPEG
```

区切り記号「itemDelimter」の変更

リンクの違いは、イメージ「dog」のファイル・ネーム（fileName）から分かりますから、最後のファイル名をそこから採り出せれば、保存ファイルに使う事ができます。例えば「http://kenjikojima.com/livecode/dog/dog3.gif」がファイル・ネームですから、最後のスラッシュ「/　slash」キャラクターの後が、保存するファイル名になります。イメージ・オブジェクトとボタンの左を揃える時に、トップレフト（topLeft）のアイテム1（item 1）アイテム2（item 2）と言う言葉を使いました。これはカンマ「, comma」で区切られた順序をアイテム・ナンバーで呼んでいます。つまりデフォルトでの区切りはカンマ「, comma」に決められているので、それをスラッシュ「/」に変更するスクリプトの「itemDelimiter　アイテム・デリミター」を使います。「item」は日本語でもアイテムと言いますし、項目と言う意味です。「delimiter　デリミター」は「範囲をさだめる」意味の「delimit」からの言葉で、日本語では「区切り記号」とも言われます。LiveCodeでは「itemDelimiter　アイテム・デリミター」とも、省略した「itemDel　アイテム・デリ」とも、どちらの形でも使う事ができます。アイテム・デリミターを、デフォルトのカンマ「,」からスラッシュ「/」に変えるのは

```
set the itemDelimiter to "/"
```
アイテム・デリミター（itemDelimiter）をスラッシュ「/」に設定しなさい。

これでアイテムを区切る記号は、スラッシュ「/」に変わって、長いファイル・ネームの最後のアイテム（the last item）から、保存するファイル名が得られます。

他のLiveCodeに類似する言語では、「itemDelimiter　アイテム・デリミター」は必ずカンマに戻しておかないと、次にカンマを使うスクリプトでエラーになってしまいますが、LiveCodeではメッセージ・ハンドラーを抜けると自動で、区切り記号（itemDel　アイテム・デリ）はデフォルトのカンマ（,　comma）に戻されます。

ここでは複雑さを避けるために全て「JPEG」でファイル形式を作る事にしますから、最後のドットの後のエクステンションは「.jpg」に統一します。ファイル・ネームの最後のアイテムを採り出して、さらにキャラクターの「（始めからの番号）1 から （最後からの番号）-5」を採り出して、その後に「.jpg」を付けます。「char」は「character」の短縮形です。

```
set the itemDel to "/"
put char 1 to -5 of the last item of the fileName of image "dog" & ".jpg"
```

```
区切りの記号（itemDel　アイテムデリ）をスラッシュ「/」に設定しなさい。
イメージ「dog」のファイル・ネーム（fileName）の
　最後のアイテムのキャラクター1から-5と「.jpg」を書き出しなさい。
```

これで、写真が切り替わるたびに違うファイル名が書き出せますから、ファイル名を「tFileName」と言うバリアブル(中身を入れ替えられるコンテナー) に入れて使います。これまでのスクリプトを纏めて、ボタン「デスクトップに写真を保存」に入れます。

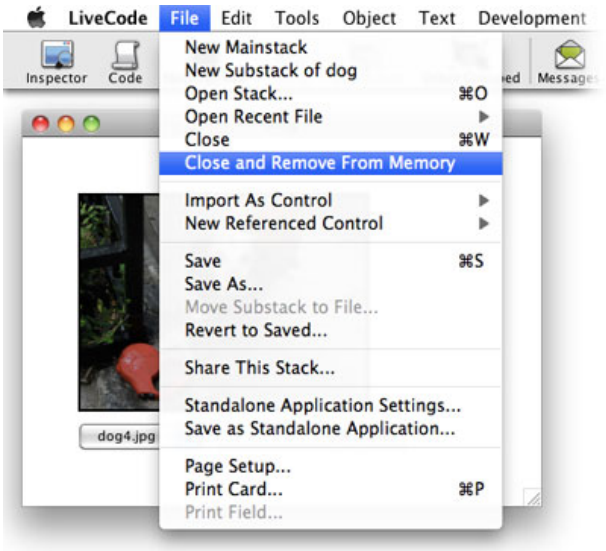
```
on mouseUp
   set the defaultFolder to specialFolderPath("Desktop")
   set the itemDel to "/"
   put char 1 to -5 of the last item of the fileName \
      of image "dog" & ".jpg" into tFileName
   export image "dog" to file tFileName as JPEG
end mouseUp
```

```
オン　マウスアップ
    デフォルト・フォルダー（defaultFolder）をデスクトップ（Desktop）に設定しなさい。
    区切りの記号（itemDel　アイテムデリ）をスラッシュ「/」に設定しなさい。
    イメージ「dog」のファイル・ネーム（fileName）の \
        最後のアイテムのキャラクター1から-5と「.jpg」を「tFileName」に入れなさい。
    イメージ「dog」をJPEGフォーマットでファイル名「tFileName」でエクスポートしなさい。
エンド　マウスアップ
```

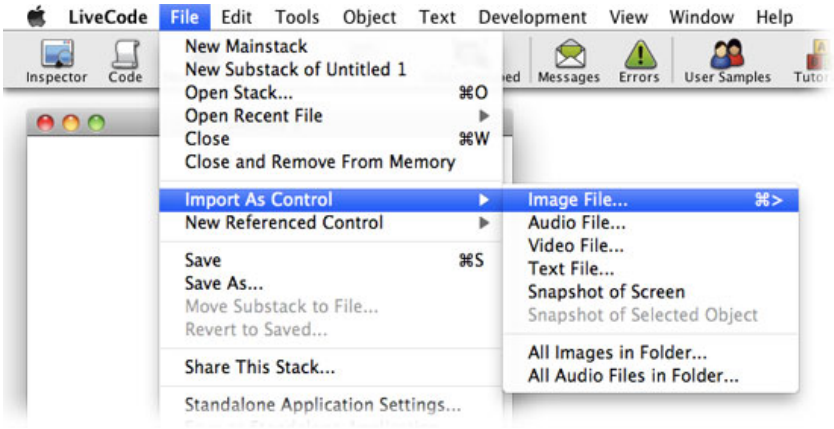
「dog1」から「dog5」までのイメージをデスクトップに保存してください。写真の画像が完全に切り替わってからボタン「デスクトップに写真を保存」をクリックしないと、正しい写真がエクスポートされません。このイメージ・ファイルのエクスポートは、同じファイル名がすでにあると上書きしてしまいます。それを避けるスクリプトは、ここでは省略します。

スタックを閉じて、新しいスタックを作る

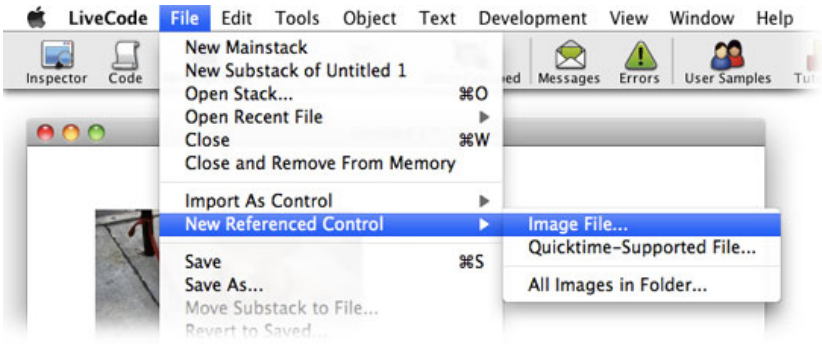
保存した5枚のイメージを使って、新しいスタックで少し実験をしてみます。スタックは「linkedImage.livecode」と言うファイル名でSave（保存）します。このスタックは閉じて、新しいスタックを作ります。必要のないスタックを閉じる場合、ただウインドウを閉じるだけにすると、LiveCode内にメモリーが残ってしまいますから、下図のように「Cloes and Remove FromMemory」を選びます。



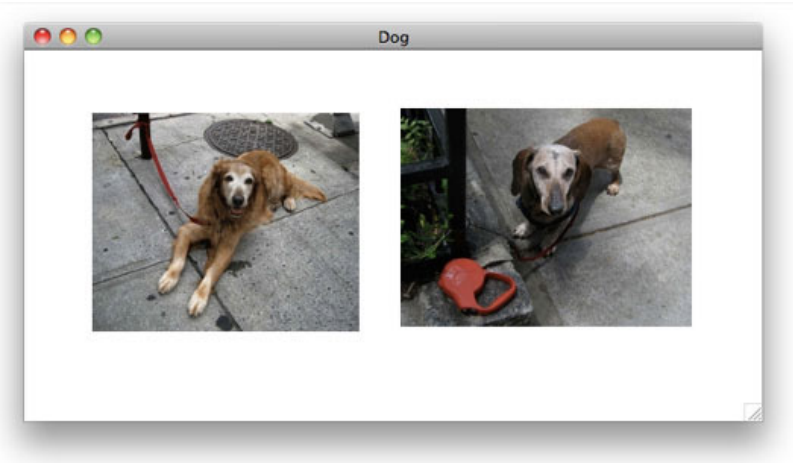
Fileメニューから「New Mainstack」を作ってください。始めに「Fileメニュー」から「Import As Control > Image File...」を選んで（下図）、デスクトップに保存したイメージ・ファイルを1つ選びます。図では「dog3.jpg」を選びました。



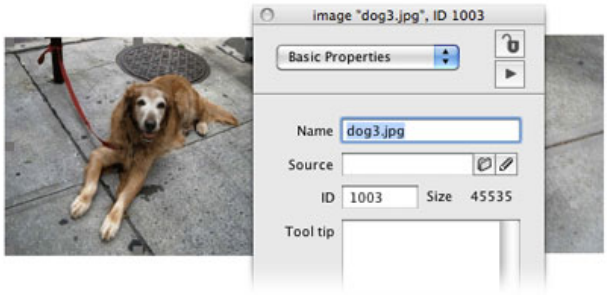
同じように「Fileメニュー」から「New Referenced Control > Image File...」を選んで（下図）、デスクトップに保存したイメージ・ファイルを1つ選びます。図では「dog4.jpg」を選びました。



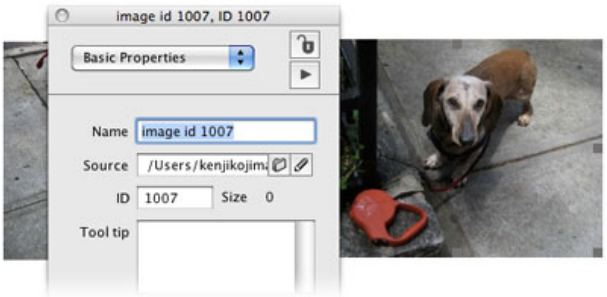
2つのイメージがカード上にありますから、適当にスタックのサイズを調整して、2つのイメージが程よい位置に来るよう配置します。スタックの名前（name）を「dog」に、タイトル（title）を「Dog」にしました。このスタックを「Save　保存」します。



始めにインポート（Import）したイメージ１（image 1）のインスペクタを見てみます。Name（名前）はオリジナル・ファイルのファイル名「dog3.jpg」となっています。「Source = fileNmae」は空白です（下図）。

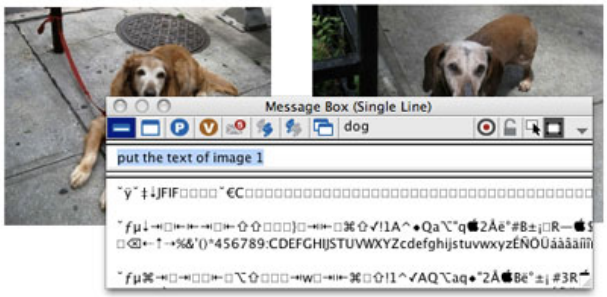


次にFileメニューのレファレンスト・コントロール（New Referenced Cotrol）からカードに置いた、イメージ２（image 2）をインスペクタで見ると、Name（名前）はイメージのIDナンバーです。「Source = fileNmae」は、ファイル・パスになっています。このことから、レファレンスト（Referenced　参照した）・コントロールと言う意味が分かります（下図）。



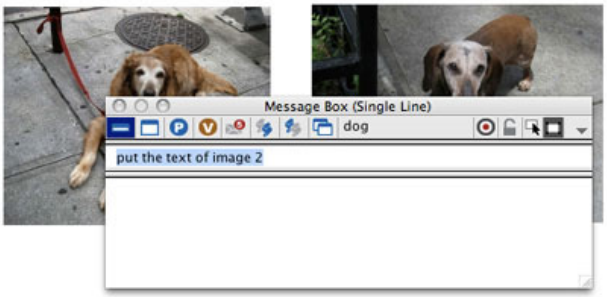

```
put the text of image 1
```

とメッセージ・ボックスから送ると、お絵描き（ペイント）の時のように、沢山の文字列が書き出されます（下図）。

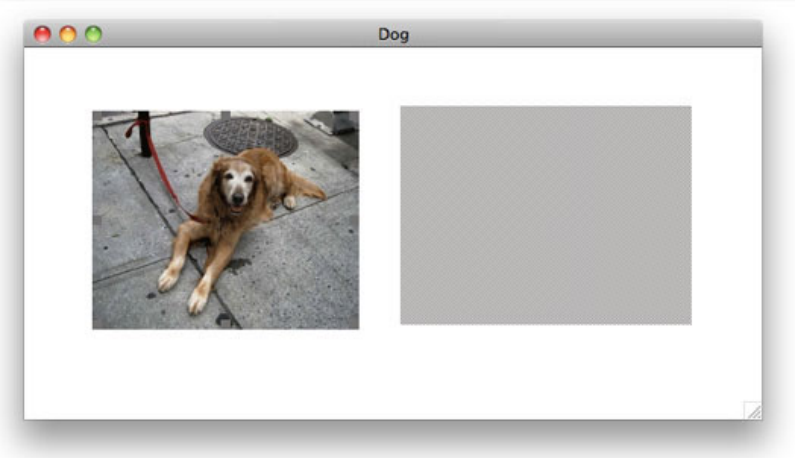



```
put the text of image 2
```

イメージ2（image 2）で、上と同じスクリプトを送ると、何も書き出されません（下図）。



Fileメニューから「CLose and Remove From Memory」で スタック「Dog」を閉じます。メモリーごとスタックを閉じました。スタック「Dog」のイメージ2（image 2）のデスクトップにあるファイル（dog4.jpg）を、どこかデスクトップでない場所に移動させてください。それではもう一度、スタック「Dog」をFileメニューから開きます。「Fileメニュー > Open Stack...」（下図）。



イメージ2は参照していただけなので、ファイルが見当たらなくなると、画像が現れません。イメージ・ファイルは、この2つの違いを理解して使い分ける必要があります。

リンクしたイメージは

```
the text of イメージ・オブジェクト
```

では、「empty　エンプティ」を返しますが、実はテンポラリーでイメージ・データ（imageData）をオブジェクトに含んでいます。 次の章はデスクトップに保存したイメージを使って、もう少しイメージの可能性を探る予定です。

Tips

- オブジェクトを作る特に、新しいオブジェクトはレイヤーの一番上に来るので、「the last」を使えば、オブジェクトのナンバーを毎回指定する必要がありません。

- リンクするイメージは、イメージ・オブジェクトのファイル・ネーム（fileNmae）を、インターネットのイメージは「http://ドメイン/ファイル名」に、ローカルのイメージは「ファイル・パス」を設定します。

- イメージ・オブジェクトをエクスポートできるファイル・フォーマットは「JPEG, GIF, PNG, BMP」です。

- LiveCodeでは「itemDelimiter　アイテム・デリミター」の設定は、ハンドラーを抜けるとデフォルトのカンマ（comma）に戻ります。

- 文字列の中をキャラクター・ナンバーで区切って短く採り出す場合「char ポジティブ・ナンバー（始めから） to ネガティブ・ナンバー（終わりから）」と言う設置もできます。

- 必要のないスタックを閉じる場合、Fileメニューから「Cloes and Remove FromMemory」を選んで、メモリーごと削除します。

この章で新しく出て来た言葉

templateImage
キーワード（keyword） 新しく作るイメージ・オブジェクトのプロパティを設定する
```
set the rect of the templateImage to 80,80,320,320
```

showborder
プロパティ（property） オブジェクトのアウトラインを表示するかしないか
```
set the showborder of the last image to true
```

threeD
プロパティ（property） オブジェクトを立体的に見せるかそうでないか
```
set the threeD of the last image to false
```

choose
コマンド（command） ペイント・ツールを選ぶ
brush
キーワード（keyword） イメージに筆で描くペイント・ツール
tool
キーワード（keyword） 現在選んでいるツール
```
choose brush tool
```

brush
プロパティ（property） ペインティングのブラッシュ・ツールの形

```
set the brush to 8
```

brushColor
プロパティ（property）ペインティングのブラッシュ・ツールの色
```
set the brushColor to blue
```

dragSpeed
プロパティ（property）dragコマンドの早さを特定する
```
set the dragSpeed to 120
```

drag
コマンド（command）マウスでドラッグしたように見せる
from
キーワード（keyword）コマンドと一緒に使われてオリジナル・ポイントを示す
```
drag from 90,90 to 300,300
```

delete
コマンド（command）指定したしたオブジェクトを削除する
```
delete image 2
```

formattedWidth
プロパティ（property）オリジナル・コンテンツの横巾の数値
```
set the width of image 1 to the formattedWidth of image 1
```

formattedHeight
プロパティ（property）オリジナル・コンテンツの天地の数値
```
set the height of image 1 to the formattedHeight of image 1
```

topLeft
プロパティ（property）オブジェクトの左上のロケーション
```
put the topLeft of button "dog"
```

bottomLeft
プロパティ（property）オブジェクトの左下のロケーション
```
put the bottomLeft of image "dog"
```

export
コマンド（command）指定したイメージをエクスポートする
```
export image "dog" to file "testImage.jpg" as JPEG
```

defaultFolder
プロパティ（property）filesやfoldersファンクションを呼び出せる特定のフォルダー
```
put the defaultFolder
```

files
ファンクション（function）デフォルト・フォルダーのファイル・リストを返す
```
put the files
```

specialFolderPath
ファンクション（function）システムに関連した特定のフォルダーのパスを返す
```
put specialFolderPath("Desktop")
set the defaultFolder to specialFolderPath("Desktop")
```

itemDelimiter

プロパティ（property）文字列を区切る特定のキャラクター　短縮形：itemDel

```
set the itemDelimiter to "/"
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

12: イメージはバイナリー・データ

この章の概略

- *ファイルを選ぶダイアログ・ウインドウを開いた時に、特定のフォルダーをデフォルトで開けたり、選べるファイルのタイプを指示したりする方法。*
- *スクリプトを使ってイメージ・ファイルをインポートする。*
- *イメージ・データ（imageData）がバイナリーと言う事。ピクセルに含まれる４つのキャラクター。ベースコンバート（baseConvert）、バイナリーの保存等初歩的な扱い。*
- *ストリング・オペライター（string operators）と、その他のオペレイター。*
- *テキストを含むイメージの保存、ネット上にあるイメージの保存。*
- *スイッチ（switch）文、ラジオ・ボタンの扱い。*

この章で使われる英単語

- choose [動詞：選ぶ]
- as [接続詞, 前置詞：...のように, ...として]
- sheet [名詞：シート, 布, 1 枚の紙]
- structure [名詞：構造]
- switch [名詞, 動詞：スイッチ, 取り替える, 切り替える]
- case [名詞：（個々の）事例, 場合]
- concatenation [名詞：連結]
- numeric [形容詞：数の]
- arithmetic [名詞, 形容詞：算数（の）]
- logical [形容詞：論理的な]
- comparision [名詞：比較]
- compression [名詞：圧縮]
- binary [形容詞, 名詞：バイナリー, 対（の）, 2 進法（の）]
- base [名詞：ベース, 土台, 位取りの基準]
- convert [動詞：変形する, 変換する]

ファイルを選ぶダイアログ・ウインドウ

前の章でメニューからイメージをインポートしましたから、今度はスクリプトでイメージをインポートします。Fileメニューから、新しくメイン・スタックを作ってください。始めにファイルを選ぶ為のコマンドを、メッセージ・ボックスから試します。

```
answer file "Choose :"
```

ファイルを選ぶウインドウが現れて、全てのファイルが見えるので、どのファイルでも選ぶ事ができます。キャンセルで閉じたら、次にこれを試してください。

```
answer file "Choose :" with specialFolderPath("Desktop")
 デスクトップ上にある「LiveCodeStudy」と言うフォルダーの中を見せたかったら
 answer file "Choose :" \
        with specialFolderPath("Desktop") & "/LiveCodeStudy"
```

「with」の後にフォルダーのパスを指定します。この場合は「specialFolderPath("Desktop")」デスクトップがデフォルトなフォルダーとして開かれます。特に指定しなくてよい場合は、始めのように省いて書きます。

```
answer file "Choose JPEG Image:" \
     with specialFolderPath("Desktop") with type "jpeg |jpg"
```

デフォルトでデスクトップを開いて、JPEGイメージ以外のファイルは選べません。フォルダーも選べますが、中を開けて行くと、やはりJPEGイメージだけが選べます。「with type」の後ろに「jpeg（ファイルタイプ）| jpg（エクステンション=拡張子）」を付けると、ユーザーは特定のファイルだけを選ぶ事ができます。さらに「gif」や「png」も選ばせたい場合には

```
answer file "Choose Image File:" \
```

```
  with type "jpeg or gif or png |jpg,gif,png"
```
最後のエクステンション（拡張子）のリストは、カンマの後ろにスペースを取りません。
フォルダーのパスは省いています。

とします。もうひとつMac OSユーザーの為に、ウインドウの上からシートが下りて来るダイアログにするには、最後に「as sheet」を付けます。

```
answer file "Choose JPEG Image:" with type "jpeg |jpg"  as sheet
```
Windowsで「as sheet」を付けても、通常のダイアログが出てくるだけです。

ダイアログについてはこれくらいにして、イメージのインポートに入ります。

スクリプトでイメージのインポート

前章のイメージの「export　エクスポート」に対して、イメージの「import　インポート」です。LiveCodeは写真、お絵描き等のビットマップ画像を、ペイント（paint）としてインポートします。

```
answer file "Choose Image:" as sheet
if it is not empty then import paint from file it
```

いろいろ前書きでダイアログについて書いた割に簡単ですね。まあ、簡単は良い事です。「it」は「answer file」で得たファイル・パスです。「it」が「empty」でない（is not）時だけ「import paint」が実行されます。

では、このスクリプトをマルチライン・メッセージから送ります。Mac OSだったら、ウインドウのタイトル・バーから、ファイルを選ぶダイアログが下りてきます。普通のダイアログにしたい時は「as sheet」を付けません。Windowsは付けておいても、通常のダイアログが出ますから、Mac OSとWindosを同時に開発する時は「as sheet」を付けたままでも問題ないです。

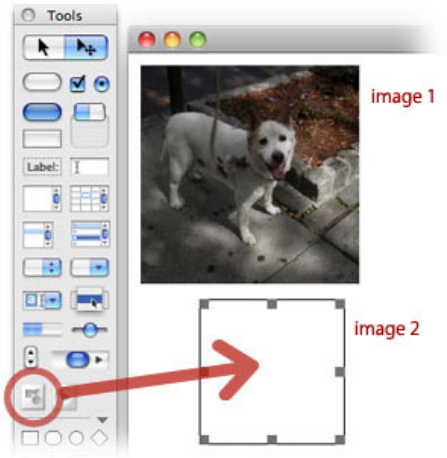
デスクトップにあるJPEGイメージを選んで、「開く（Open）」ボタンをクリックすると、カードの真ん中にイメージがインポートされます。イメージ・ファイルはどれを選んでも良いです。ここでは「dog2.jpg」を選びました。エディット・ツールで、インポートしたイメージをカードの左上隅に移動させます。このイメージは、メニューからコントロールとしてインポートしたイメージと同じです（Import As Control）。「import paint」を使うと、「GIF, JPEG, PNG, BMP, XWD, XBM, XPM, PBM, PGM, PBM」のビットマップ画像はLiveCodeにインポートできます。「PICT」もインポートできますが、Mac OSのみで可能です。

インポートしたコントロール画像のフォーマット（圧縮形式）を知るには、プロパティの「paintCompression」を使います。

```
put the paintCompression of image 1
```
イメージ1（image 1）のペイント・コンプレッションを返しなさい。

インポートしているイメージが「JPEG」ですから、当然「jpeg」が返されます。

イメージ・データ「imageData」をセット

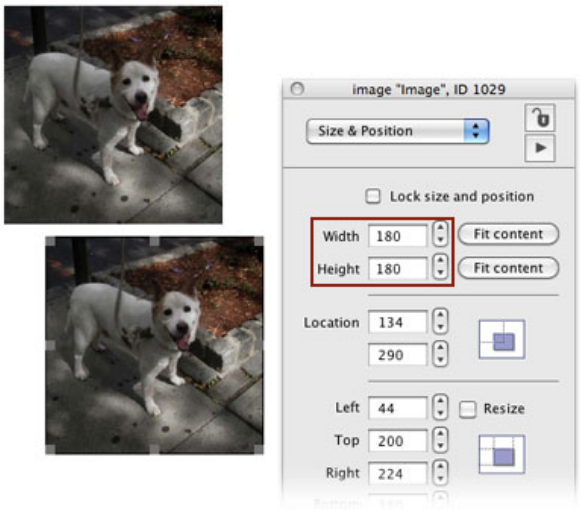


次にツール・パレットからイメージ（image 2）を置いてください（上図）。前の章「お絵描き（ペイント）」では「put the text of image 1」を使って、イメージのテキストを他のイメージに入れて、まったく同じイメージを作りました。ここではイメージ・データ（imageData）を作ったイメージ２（image 2）にセット（set）します。イメージ２（image 2）のサイズは、デフォルトのままにしておいてください。イメージデータ（imageData）はプット（put）ではなく、イメージ２（image 2）をイメージ１（image 1）にセット（set）します。

```
set the imageData of image 2 to the imageData of image 1
```

イメージ２（image 2）のイメージデータ（imageData）を
イメージ１（image 1）のイメージデータ（imageData）に設定（set）しなさい。

前の章「お絵描き（ペイント）」のテキストをプットした時とは違って、サイズは変わらないでグリッチ（欠陥）のようなイメージになってしまいます。イメージデータ（imageData）には、イメージ・フォーマット（圧縮情報）やサイズ等のデータは含まれていません。元のイメージが持っているピクセルのデータだけを返すプロパティです。ですからサイズの違うイメージ・オブジェクトにセットすると、画像が崩れてしまいます。



イメージ２（image 2）をオリジナルのイメージ１（image 1）と同じ左右天地にして、スクリプトを送ると今度は同じ画像を結びます。

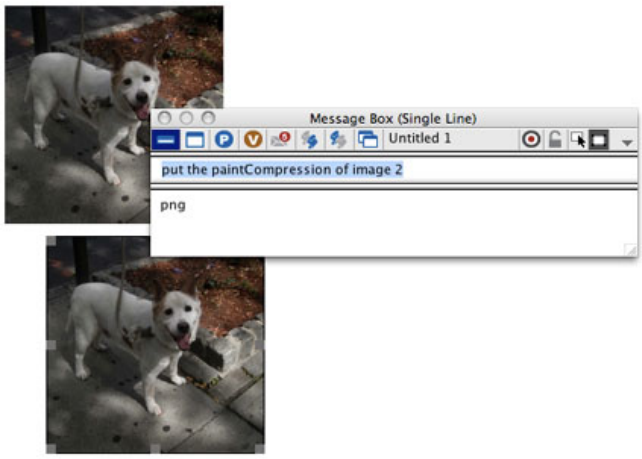
```
set the imageData of image 2 to the imageData of image 1
```

イメージ２（image 2）のイメージデータ（imageData）を
イメージ１（image 1）のイメージデータ（imageData）に設定（set）しなさい。

イメージ２（image 2）のペイント・コンプレッション（paintCompression　画像の圧縮情報）を見てみます。

```
put the paintCompression of image 2
```

イメージ２（image 2）のペイント・コンプレッションを返しなさい。



ペイント・コンプレッション（paintCompression）は「png」が返されます。イメージ・データをカード上で他のイメージ・オブジェクトにセットすると、イメージの形式は画像のクオリティを損なわずに、カラーの加工やアルファ・チャンネルが扱える、PNG画像になります。

１ピクセルの中の４つの情報

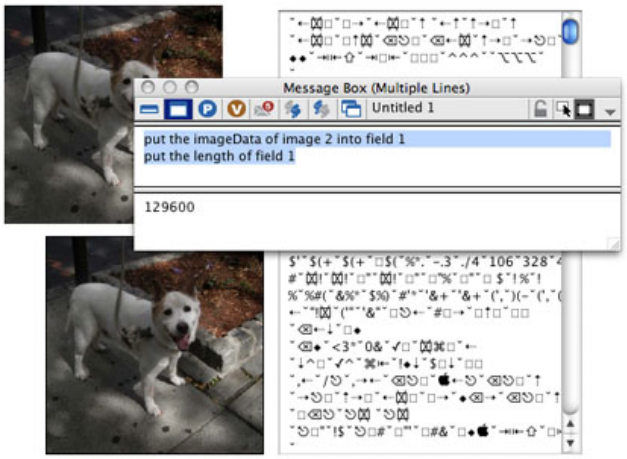
ここでは基礎的なピクセルの知識だけに留めて、あまりデープな取り扱いの方法には触れません。イメージ２は横に180個のピクセルが、縦には180列並んで全体の画像を形成しています。

つまりピクセル数は「180 x 180 = 32400ドット」です。ひとつのピクセルには実は４種類の情報が書き込まれていて、画像全部のピクセル情報を、イメージ・データ（imageData）プロパティで呼び出す事ができます。ですから呼び出せるイメージ２のイメージ・データの情報量は、ピクセルの総数 x ４（180 x 180 x 4 = 129600）書き込まれています。

カード上にスクローリング・フィールド（Scrolling Field）「field 1」をツール・パレットから作ってください。そのフィールドにイメージ・データを書き出して、イメージ・データの文字数をメッセージ・ボックス下段に書きます。下のスクリプトをメッセージ・ボックスから送ります（下図）。

```
put the imageData of image 2 into field 1
put the length of field 1
```

イメージ２（image 2）のイメージ・データ（imageData）をフィールド１（field 1）に入れなさい。
フィールド１（field 1）の文字数を返しなさい。



メッセージ・ボックスに書かれた数字が、総ピクセル数の4倍の数だと分かります。そしてその文字数の情報がイメージ・データ（imageData）としてフィールドに入っています。私たちはフィールドに入っているこの画像情報を、バイナリー・データと呼んでいます。バイナリー(binary) とは２進数の意味で、私は始め何故この文字記号の並んでいるイメージ・データを、２進数（バイナリー）と呼ぶのか理解できませんでした。確かに普通のテキストようには読めないけれど、これもテキストではないかと思っていたのです。と言うよりバイナリーだから「０と１」と言う表記ではないのかと思っていたからです。それはこの章の後半で説明します。

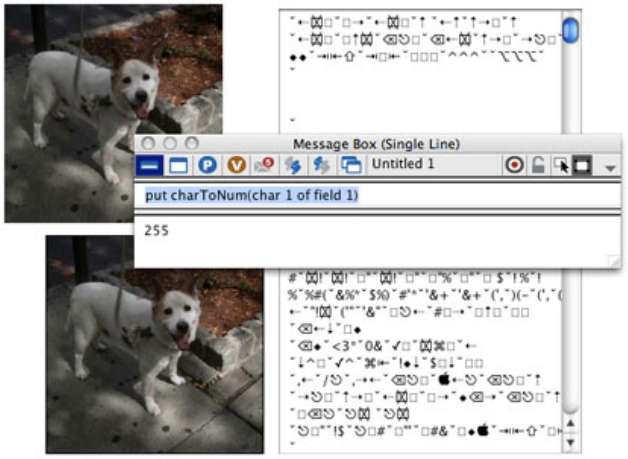
フィールドの中にある文字列のキャラクター1〜4（char 1 to 4）までが、イメージ２の最上列いちばん左端のピクセルで、その右隣のピクセルはキャラクター5〜8（char 5 to 8）です。そのひとつひとつのキャラクターにピクセルの画像情報が入っています。画像情報とは具体的には、アルファ・チャンネル（透明度）と、赤（Red）緑（Green）青（Blue）の色彩情報です。

今はテキスト・フィールドに文字や記号が並んでいるのが見えますが、この文字のひとつひとつには「0から255」までの数字が割り振られています。10章に書いたASCII（アスキー）を思い出してください。ASCII（アスキー）は、ヨーロッパ言語で使われるアルファベット、数字、記号のキャラクターに数字を割り当てた文字コードで、これを使うと「0から255」までの数を1文字で表す事ができます。

もうひとつ色彩については、6章「RGBカラーの基礎知識」に少し書きました。RGBそれぞれに「0から255」までの数字を含んでいて、その割合で色彩を表しています。RGBが「0,0,0」なら「黒」、RGBが「255,255,255」なら「白」です。さてそこでASCII（アスキー）の時に話した、キャラクターを数字に変換するファンクション「charToNum」を覚えているでしょうか。今フィールドの中に入っているイメージ・データは、キャラクター毎にファンクション「charToNum」を使って「0から255」に変換できます。

```
put charToNum(char 1 of field 1)
```

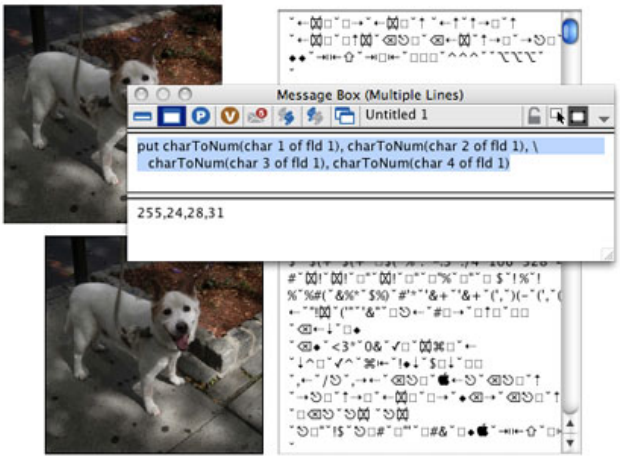
フィールド１（field 1）のキャラクター１（char 1）を
ASCII（アスキー）番号に変換して書き出しなさい。



ピクセルに含まれる情報の最初のキャラクターは、アルファ・チャンネル（透明度）です。アルファ・チャンネルが「255」と言う意味は、そのピクセルは完全にオペーク（不透明）を意味します。続く３つのキャラクターは「RGB」の順になっています。キャラクターの１から４までを、カンマで接続させて書き出します。

```
put charToNum(char 1 of fld 1), charToNum(char 2 of fld 1), \
   charToNum(char 3 of fld 1), charToNum(char 4 of fld 1)
```

フィールド１（fld 1）のキャラクター１（char 1），
フィールド１（fld 1）のキャラクター2（char 2），
フィールド１（fld 1）のキャラクター3（char 3），
フィールド１（fld 1）のキャラクター4（char 4）を
ASCII（アスキー）番号に変換して書き出しなさい。



「255,24,28,31」と書き出されました。書き出したイメージ・データの最上段左端のピクセルの情報は、「255」完全に不透明で続くRGBの「24,28,31」は、ほとんど黒に近いピクセルと言う意味です。プログラミングが書けると言う事のひとつは、このようにして１つ１つのピクセル情報を取り出して、透明にしたり、色彩を変えたり等、大容量のデータをコンピュータで高速に処理させる事ができるという事でもあります。

文字列の連結「String Operator」

ここで上のスクリプトで使った「カンマ　comma , 」の用法について書き足しておきます。LiveCodeではコンカチネーション（concatenation　連結）と呼んでいるオペレイター（operator　操作記号）が３つあります。コンカチネーションはストリングス（strings　文字列）を連結させる記号で、別の呼び方ではストリング・オペライター（string operators）とも言って、すでに何度も使っている「&」と、後の２つが「&&」と「,」です。

「&記号」は文字列のチャンクをスペースなしで繋げます。例えば

```
"Down " & "the " & "Rabbit-Hole"
```

「&&記号」は文字列のチャンクをスペースで繋げます。例えば

```
"Down" && "the" && "Rabbit-Hole"
```

結果はどちらも「Down the Rabbit-Hole」です。

「,（カンマ）」は、普通には区切りの意味ですが、文字列を繋ぐ連結記号としても使われて、その結果として書き出された連結文字列のカンマ記号は「itemDel　区切り記号」の役割を果たします。意味合いからは「& comma &」又は「& "," &」と同じです。例えば、

```
put "Alice" , "Sister" , "White Rabbit" , "Mock Turtle"
```

結果は「Alice,Sister,White Rabbit,Mock Turtle」になります。

LiveCodeのオペレイター（operator 操作記号）は、大きく分けて計算式（数字だけでなく文字

列も式と考えて）を扱うヌーメリック・オペレイター（numeric operators　計算式操作）と、真か偽かを判断するロジカル・オペレイター（logical operator　論理操作）があります。 基本的なオペレイターを下に書きました。

ヌーメリック・オペレイター（numeric operators　計算式操作記号）

「+」「-」「*」「/」「div」「mod」　*（arithmetic operators　算数操作）*

「&」「&&」「,」　*（string operators　文字列連結操作）*

「( )」　*(grouping operator　グループ化操作）*

ロジカル・オペレイター（logical operators　論理操作記号）

「=」「<」「<=」「>」「>=」　*（comparision operators　比較操作）*

「is a」「is not a」　*（data type operators　データ・タイプ操作）*

「is within」「is not within」　*（geometry operators　幾何操作=ある範囲内かどうか）*

「and」「or」「not」　*（basic logical operators　基礎論理操作）*

バイナリー「2進数」と
ベースコンバート「baseConvert」

イメージ・データ（imageData）はバイナリーで書き出されると上に書きました。少しその説明をしましょう。ここにまったく透明感のない不透明な３つのピクセルが並んでいる、小さな画像があるとします。下の表を見ながら読んでください。

始めのピクセルは「Red」、次のピクセルが「Green」、最後は「Blue」のピクセルです。いちばん始めの「Redピクセル」には、キャラクター１（char 1）から４（char 4）までのデータが含まれています。「char 1」はアルファ（透明度）の情報、「char 2」は「Red 赤」の情報、「char 3」は「Green 緑」の情報、「char 4」は「Blue 青」の情報を持っています。次の「Greenピクセル」には、キャラクター５（char 5）から８（char 8）までのデータが含まれています。最後の「Blueピクセル」には、キャラクター９（char 9）から１２（char 12）までのデータが含まれています。

不透明な「Redピクセル」の、透明度の情報キャラクター１（char 1）をASCIIにすると「255」です。キャラクター２（char 2）からキャラクター４（char 4）までのASCIIに変換したRGB値は「255,0,0」で、「Red」が最高値、後の「Green, Blue」はゼロですから、視覚的には赤（Red）に見えています。

	Red ピクセル				Green ピクセル				Blue ピクセル			
	アルファ	R	G	B	アルファ	R	G	B	アルファ	R	G	B
キャラクター	1	2	3	4	5	6	7	8	9	10	11	12
ASCII	255	255	0	0	255	0	255	0	255	0	0	255
バイナリー	11111111	11111111	00000000	00000000	11111111	00000000	11111111	00000000	11111111	00000000	00000000	11111111

２進法は「0」と「1」だけで数値を表す方法で、英語ではバイナリー（Binary）と言います。少しだけ基礎的な事に触れておきます。２進数の最小の単位の１ビット（1 bit）は、１桁の「0」と「1」だけで表現できる数で、普通私たちが使っている１０進数で表現している「0」と「1」と同じ意味です。

２進数は「0」と「1」を２桁使うと（2 bits）では、「00 = 0」「01 = 1」「10 = 2」「11 = 3」（カッコ内の左の２桁が２進数、イコール記号の右が１０進数です）となります。このように１が桁の全てに並んで使える文字が無くなると、桁数を上げて次の位に行きます。そのように計算をして行くと、１０進数の「255」は８桁の２進数「11111111」となります。

８桁の２進数を８ビット（8-bits）と言って、画像を構成している１文字（charactor）は、８ビットのメモリー・サイズで作られています。つまり４文字で構成されるピクセル１つのメモリー・サイズは、８ビットが４文字で３２ビット(32-bits) と言う事です。イメージ・データ（imageData）で得られるデータは、見かけ上は文字列ですが、実際はこのように、生のバイナリーのデータなのです。

普通の英語のようなコンピュータ言語でも、ちょっと筋道を探って行くと、こんな生なバイナリー・データに出会えるなんて、ちょっとドキドキしませんか？

LiveCodeでは、位取りの異なる数を別な位取りに変換する「baseConvert　ベースコンバート」と言うファンクションを使うことができます。「base　ベース」は基準とする位の事で１０進法では「10」、２進法では「2」、１６進法では「16」を指します。

```
baseConvert(255,10,2)
```
「11111111」を返します。
baseConvert(変換したい数，オリジナルのベース，変換したいベース)

```
baseConvert(255,10,16)
```
「FF」を返します。　１６進数はウェブのhtml書類などで使うカラーの表示法です。

イメージのテキストを保存

イメージ・データ（imageData）は画像の持つピクセルのバイナリーな情報でしたが、イメージのテキストは画像のサイズや圧縮形式等も含むバイナリーな情報です。イメージのテキストを保存する場合、１０章のテキスト・フィールドから普通のテキストを保存する方法とは違う、バイナリーの保存方法を取る必要があります。英文のテキストの保存をもう一度見てみます。英文テキストはフィールド１（field 1）にあって、デフォルトのファイル名は「Alice.txt」にしています。

```
ask file "Please name the file:" with "Alice.txt"
put field 1 into url ("file:" & it)
```
「it」はファイルを保存する場所の、ファイル名を含むパスです。

「file:」と言う言葉に注目してください。普通のプレインな英文テキストは「file:」を使います。バイナリー・データの保存は「binfile:」にします。このスクリプトでイメージ１のサイズや圧縮情報も含むデータを、ファイル・パス「it」に保存します。

```
put the text of image 2 into url ("binfile:" & it)
```
イメージはバイナリーなので「"binfile:"」とします。

基本的な構文としては、これでイメージのバイナリー・データをファイル・パスに保存できま

すが、デフォルトで付ける名前のエクステンション（拡張子）を確認した方が良いですね。上で使ったように画像のフォーマットを知るには、プロパティの「paintCompression」で得る事ができます。

```
put the paintCompression of image 1

put the paintCompression of image 2
```

この２行のスクリプトを別々にメーッセージ・ボックスから送ると「image 1」では「jpeg」、「image 2」では「png」が返されます。「jpeg」のエクステンション（拡張子）は「.jpg」も「.jpeg」も使えるので、プロパティの「paintCompression」で得られた「jpeg」でも良いのですが、アプリケーションとしては、エクステンション（拡張子）を統一した方が良いかもしれません。メニュー・ボタンのスクリプトの時に「switch」を取り除いて作ったので、コントロール・ストラクチャー（control structure）の「switch」の学習を兼ねてエクステンション（拡張子）を得るスクリプトを作ります。今カード上には「gif」写真はありませんが、「gif」も加えたスクリプトにします。

スイッチ「switch」文

スイッチ（switch）は、いくつかの可能性のあるバリュー（value）から、該当するものを選ぶ構文で、得られる結果は「if」とも似ています。「if」の場合は、該当しなければ、次々に「もしそうでなければ」「もしそうでなければ」と、ドンドン中に入ってチェックして行きます。「switch」は複数の可能性を並列に用意して、「case　ケース」を使って、「この場合は」「この場合は」と上から順にチェックして、該当するバリュー（value）に当たった時、その中に用意したスクリプト（ステートメント）を実行します。この下に書いたファンクションは、「paintCompression」で得る事が予想される、３つの可能性「jpeg,png,gif」がどれかだったら、このステートメントを実行しなさいというスクリプトです。「--」の後の日本語はコメント（説明）ですから、スクリプトには書き込めません。

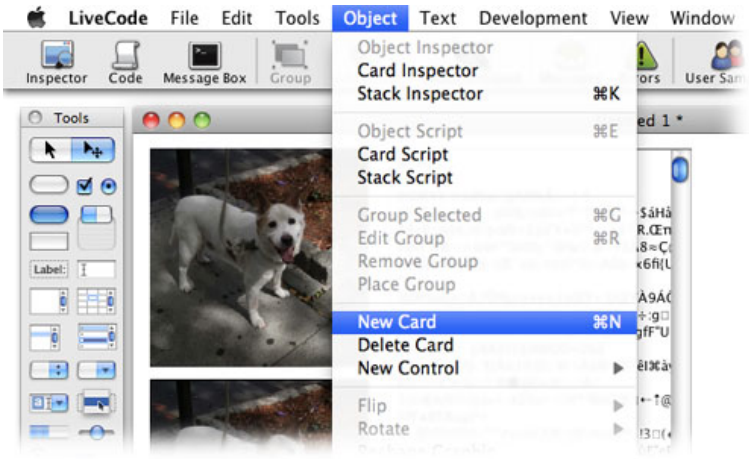
```
function getExtension pImageNum
   put the paintCompression of image pImageNum into tPaintCom
   -- パラメータ（pImageNum）は、イメージの番号（または名前）
    -- そのイメージのペイント・コンプレッションをtPaintComに入れる

   switch tPaintCom  -- tPaintComはスイッチの振り分けで該当する語
      case "jpeg"
         -- 該当する語が「jpeg」の場合
         return ".jpg"
         break   -- 上の「return」でバリューを返えすので、
             -- 他の「case」をチェックしないように「break」でswitchから抜ける
      case "gif"
      case "png"
         -- 該当する語が「gif」でも「png」の場合でも
         return "." &  tPaintCom
         break   -- 他の「case」をチェックしないようにswitchから抜ける
   end switch    -- 必ず end switch で終わる

end getExtension   -- ファンクション終わり
```

ファンクション「getExtension」の赤文字にした所が「switch文」です。他の機械語に近いコンピュータ言語の影響で、少し普通の英語文でない語順ですが、基本で使っている言葉は「switch」「case」「break」だけの、ごく簡単なものです。始めの「switch」の後にある語（ここでは「tPaintCom」）は「スイッチ・エクスプレション　switch expression」と呼んで「case」に付いている語と同じ場合、その「case」のステートメントを実行します。ステートメントが実行されたら、他の「case」に行かないように、必ず「break」で「switch」から抜けなければいけません。ここでは「getExtension」と言うファンクションの中ですから「return」でバリュー（value）を返します。例えば「mouseUp」の中のような場合は、コマンドを実行する事もできます。このサンプルでは、該当する言葉が明確になるように３つの「case」を並べて使っていますが、「case "gif"」と「case "png"」は纏めてその他とする「default」と置き換える事もできます。

イメージ１（image 1）のテキストを保存するボタンのスクリプトは、下記のようになります。ファンクション「getExtension」に入るパラメータは、この場合「1」です。ボタンの中に全部のスクリプトが入るとして、ファンクションは「private」にしています。

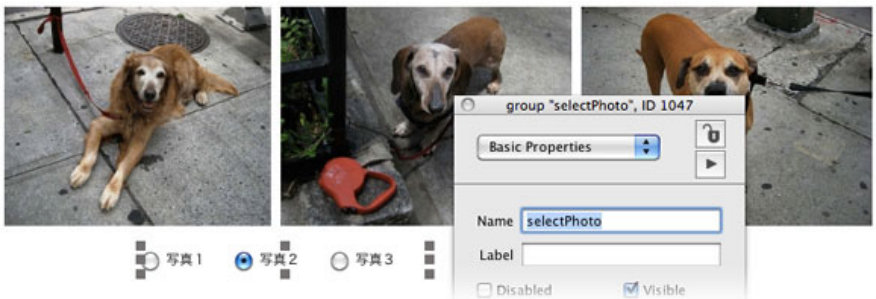
```
on mouseUp
   ask file "Please name the photo:" with "myPhoto" & getExtension(1)
   put the text of image 1 into url ("binfile:" & it)
end mouseUp

private function getExtension pImageNum
   put the paintCompression of image pImageNum into tPaintCom
   switch tPaintCom
      case "jpeg"
         return ".jpg"
         break
      case "gif"
      case "png"
         return "." &  tPaintCom
         break
   end switch
end getExtension
```

幾つかの選択から１つを選ぶラジオ・ボタン



イメージ・オブジェクトが３つあった場合に、保存するどれか１つを選ばせるラジオ・ボタンを作ります。「Objectメニュー」から「New Card」を作って（上図）、新しいカードの上に「Fileメニュー＞Import As Control＞Image File...」で、「dog3.jpg」「dog4.jpg」「dog5.jpg」を、適当な配置にインポートします。リンクしたイメージ（Referenced Control）を間違えてインポートしないでください。カード上にツール・パレットからラジオ・ボタン（Radio Button）をドラッグ＆ドロップして、レイベル（Label）は「写真１」とします。（下図）。

ラジオ・ボタン（Radio Button）を全部で３つ作って、レイベルは「写真１」「写真２」「写真３」としたら、「Objectメニュー＞Group Selcted」でグループ化して、グループの名前（name）は「selectPhoto」にします（下図）。



グループ化したラジオ・ボタンの右に、レイベルを「選んだ写真を保存」としたボタンを置きます（下図）。ラジオ・ボタンをテストしてください。グループ化されたラジオ・ボタンは、スクリプトを書き込まなくても、１つがハイライトになると（ボタンが選ばれると）、他はハ

イライトがオフになります。グループ化されたラジオ・ボタンは「the hilitedButton of グループ名」で、選ばれたボタンのグループ内でのレイヤー・ナンバーを返します。グループ内で一番下にあるボタンが 1 です。


```
put the hilitedButton of group "selectPhoto"
```

グループ「selectPhoto」の選ばれた（ハイライト）ボタンの番号を返しなさい。
この図では「2」を返します。

これを使ってボタン「選んだ写真を保存」のスクリプトにします。

```
on mouseUp
   put the hilitedButton of group "selectPhoto" into mySelection
   put the short name of image mySelection into photoName
   ask file "Please name the photo:" with photoName
   put the text of image mySelection into url ("binfile:" & it)
end mouseUp
```

オン　マウスアップ
グループ「selectPhoto」のハイライト・ボタンの番号を「mySelection」に入れなさい。
イメージ「mySelection」のショート・ネームを「photoName」に入れなさい。
「Please name the photo:」のウインドウ・タイトルで、デフォルトのファイル名を 「photoName」にした、ファイル保存のダイアログを開けます。
イメージ「mySelection」のテキストを「it」で受けたファイル・パスに入れなさい。
エンド　マウスアップ

グループ「selectPhoto」のハイライト・ボタンの番号を、バリアブル「mySelection」に入れます。「mySelection」の中身は数字の「1」か「2」か「3」になります。２行目の「image mySelection」はイメージ・ナンバーになります。もし「mySelection」に「1」が入ったら、「image mySelection」は「image 1」と言う事になります。Fileメニューの「Import As Control」でイメージをインポートしてるので、イメージ・オブジェクトの名前（name）は、「dog3.jpg」のように元のファイル名がそのまま使われているので、その名前を使います。

```
put the name of image 1
```

とすると、「image "dog3.jpg"」が返ってきます。スクリプトには「dog3.jpg」だけが必要で、「image 」とクオートは入りませんから「the short name of 」を使います。これで３行目の「photoName」にオブジェクトの名前だかが入ります。後は上でやっている事の繰り返しです。ダイアログを開けた時にデフォルトのファイル名が「photoName」になります。

追加情報：
「hilitedButton」はグループ内のハイライトになっているボタン・ナンバーを返しますが、
「hilitedButtonName」はハイライトになっているボタンの名前（name）を返します。

インターネットのイメージを保存

最後にインターネットのイメージを書いておきます。前章のように、イメージ・オブジェクトにリンクしたイメージをエクスポートすると、インターネットにあるオリジナルのサイズや圧縮情報が失われてしまいます。新しいスタックでも、ニューカードでも良いですが、イメージ・オブジェクトを１つ作ります。イメージ１（image 1）に「gif」イメージをリンクさせる事にしました。

```
set the fileName of image 1 to "http://kenjikojima.com/livecode/dog/dog3.gif"
```

これでインターネットのイメージがオブジェクトにリンクされます。以下のスクリプトは、今までにイメージで学習した事柄の総集編みたいなモノです。

```
on mouseUp
   put the fileName of image 1 into tFileName
   set the itemDel to "/"
   put the last item of tFileName into tDefaultName
   ask file "Please name the photo:" with tDefaultName
   put url(tFileName) into url ("binfile:" & it)
end mouseUp
```

２行目、リンクしたオブジェクトの「ファイル・ネーム　fileName」は、リンクしているアドレスです。「tFileName」にはそのアドレスが入ります。３行目、ファイル・ネームは「スラッシュ　/」を区切りにしている文字列ですから、アイテムデリミター（itemDelimiter）を「スラッシュ　/」に設定します。４行目、ファイル・ネームのラスト・アイテムを、ファイルを保存するダイアログを開いた時の、デフォルトの名前（tDefaultName）に入れます。５行目、「ask file 」でダイアログを開いて「it」に保存する場所のパスを入れます。６行目（最後から２行目）をもう少し詳しく書くと

```
put url("http://kenjikojima.com/livecode/dog/dog3.gif") \
     into url ("binfile:" & it)
```

となります。

Tips

- ファイルを選ぶダイアログのコマンドに、ファイル・タイプとエクステンションを記述しておけば、ユーザーは必要なファイル・タイプだけを選べます。

- Mac OSのシート・ウインドウを表示するのは、「answer file ""」の後に「as sheet」を付けます。Windowsは「as sheet」を付けたままでも普通のダイアログを表示するので、Mac OSとWindwosを同時に開発する時は、削除しなくても問題ありません。

- カンマ（comma）記号は、スクリプト中で文字列を連結させる記号としても使用できて、結果は区切り記号（itemDelimiter）としての役割を果たします。

- １０進数を２進数や１６進数に変換したり、逆に１０進数にしたりする場合には「baseConvert」と言うファンクションが使えます。

- プレインな英文テキストの保存には「url ("file:" & パス）」。バイナリー・データの保存には「url ("binfile:" & パス）を使います。LiveCodeでは日本文（ユニコード・テキスト）もバイナリーとして保存します。

この章で新しく出て来た言葉

answer file
コマンド（command）ユーザーがファイルを選べるダイアログを表示する
シート・スタイルで表示する「as sheet」はMac OSでのみ有効
```
answer file "Choose :"
answer file "Choose File:" as sheet
```

answer file with type
コマンド（command） ユーザーが選べるファイル・タイプのダイアログを表示する
```
answer file "Choose Image File:" \
      with type "jpeg or gif or png |jpg,gif,png"
```

import
コマンド（command）イメージ（ペイント）、オーディオ等をカードにインポートする
```
import paint from file it
```

paintCompression
プロパティ（property）イメージの圧縮形式
```
get the paintCompression of image "myPhoto"
```

imageData

プロパティ（property）イメージ・オブジェクトを構成するバイナリー・データ

```
set the imageData of image 2 to the imageData of image 1
```

baseConvert

ファンクション（function）位取りの違う数値を他の位取りに変換する

```
baseConvert( [変換したい数字],
   [変換したい数字のベース（何進法）], [結果の数字のベース（何進法）] )
baseConvert(255,10,2)
baseConvert(255,10,16)
```

binfile

キーワード（keyword）バイナリー・ファイルをローカルに保存する際URLに示すタイプ

```
put the text of image 2 into url ("binfile:" & it)
```

switch

コントロール・ストラクチャー（control structure）

幾つかの可能性のあるバリュー（value）を並べて選ばれたバリューのステートメントを実行する

```
switch [スイッチ・エクスプレション]
    case {バリュー1}
        [実行するステートメント]
        break
    case {バリュー2}
        [実行するステートメント]
        break
    default
        [上に並んだcase以外で実行するステートメント]
end switch
```

hilitedButton

プロパティ（property）グループ内のハイライトになっているボタンの番号

```
put the hilitedButton of group "selectPhoto" into mySelection
```

hilitedButtonName

プロパティ（property）グループ内のハイライトになっているボタンの名前（name）

```
put the hilitedButtonName of group "groupBtns" into selectedBtnName
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

13: マルチメディア 1

この章の概略

- *座標を使ってオブジェクトの位置を移動させるアニメーションや、マウスでオブジェクトを移動させるゲームの初歩的なスクリプト。*
- *オーディア・クリップとビデオ・クリップ、GIFアニメの操作方。パレット・ウインドウ。*
- *アニメーション・エンジンの紹介。*

スクリプトを使った基本的なアニメーションやサウンド、ビデオの扱い等について書きます。ゲームの初歩としても必要な事柄です。

この章で使うサウンド、ビデオのマテリアルをダウンロードしてください。

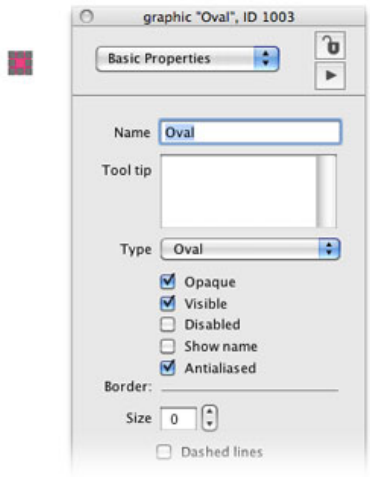
http://kenjikojima.com/livecode/download/multiMedia13.zip

この章で使われる英単語

- move [動詞：動く, 移動する]
- speed [名詞：速さ, 速度]
- grab [動詞：ぎゅっと掴む]
- tip [名詞：チップ, 心付け, 助言, 秘訣, ヒント]
- audio [名詞, 形容詞：音声（の）]
- clip [名詞：切り取り, クリップ]
- palette [名詞：パレット, 道具箱]

座標のアニメーション

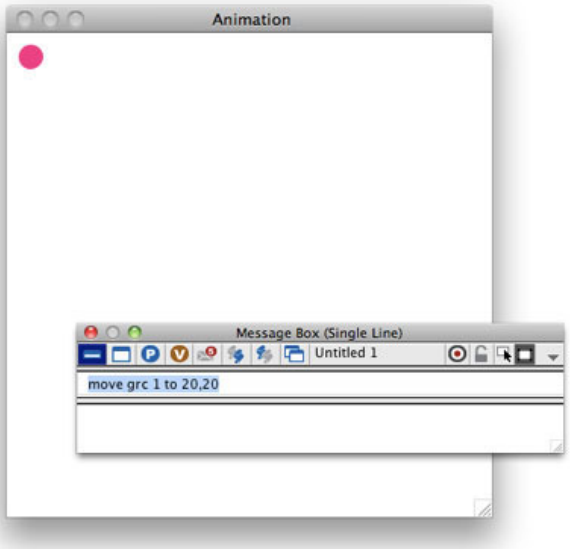
座標を使ってオブジェクトを移動させる「move」のアニメーションです。Fileメニューからニュー・メインスタックを作ります。タイトルは「Animation」にしました（このサンプルでは、タイトルはなくても差し支えありません）。ツール・パレットからカード上ほぼ中央に、小さなグラフィック（graphic, grc）のオーバル（oval）を作ります。何か色を付けた方が楽しそうなので、インスペクターでオペーク（opaque）にして、バックグラウンドカラー（backgroundColor）はストロベリーにしました。



下のスクリプトをメッセージ・ボックスから送ります。カードには作ったオーバル（oval）のグラフィックが 1 つあるだけなので、グラフィックのナンバーは 1（graphic 1）です。

```
move graphic 1 to 20,20
```

グラフィック１を20,20に動かしなさい。



そう早くもなく遅くもないスピードで、カードの左上のロケーション（the loc of graphic 1）に動いて行きます。現在はデフォルトのスピードです。オブジェクトが移動するときのスピード「moveSpeed　ムーブスピード」を聞いてみます。

```
put the moveSpeed
```

デフォルトの「moveSpeed　ムーブスピード」は「200」です。今度はスペードを変えてカードの中央にオーバルを移動させて、またスピードを変えて「20,20」に戻します。こういう場合の中央の座標はカード上のローカル座標ですから、「the location of this card」を使います。もし「the loc of this stack」とすると、デスクトップ上のグローバル座標になってしまいますから注意してください。

```
set the moveSpeed to 400
move graphic 1 to the location of this card
set the moveSpeed to 80
move graphic 1 to 20,20
```

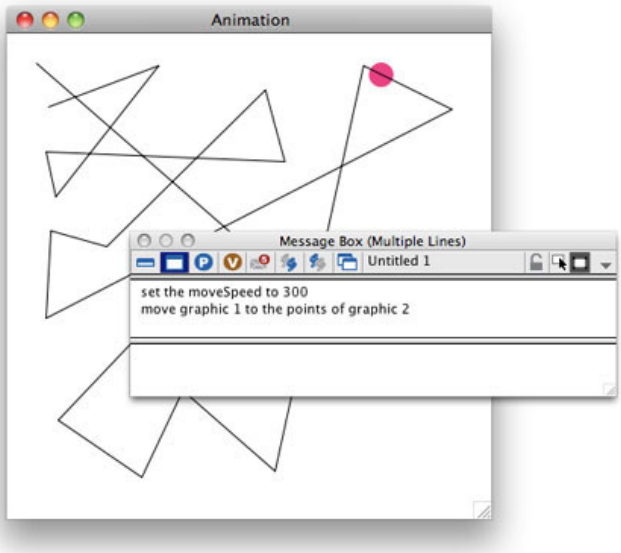
ムーブスピード（moveSpeed）は「ゼロから65535」までの整数をセットできます。動かせるオブジェクトはグラフィックとは限りません。カードに置けるコントロールでしたら、イメージでもボタンでも動かせます。「move　動け」と言う普通の感覚の言葉を使っていますが、「set the location of オブジェクト to x値,y値」としても同じ移動ができます。ただしオブジェクトのロケーションの設定では「moveSpeed　ムーブスピード」は働きません。

以上が最も基本的な、座標を使ったアニメーションです。次にそのバリエーションのフリーハンド・ポリゴン（Freehand Polygon）を使ったアニメーションをやってみましょう。何でも良いですから、適当にカード上にフリーハンド・ポリゴン（Freehand Polygon）を描きます。フリーハンド・ポリゴンはクリックした地点がポイントとなってラインを描き、最後はダブルクリックで描き終わります。作ったフリーハンド・ポリゴンは「graphic 2」です。フリーハンド・ポリゴン（Freehand Polygon）には「points」と言うプロパティがある事を思い出してください（06: グラフィック、カラー）。

```
put the points of graphic 2
```

グラフィック２（grc 2）のポインツ（points）座標を返しなさい

ポイント数だけの座標が、リターン（return, cr）をデリミッター（delimiter)として書き出されます。これを使ってXY座標と言う数学的な値でない、目に見えるポイントにオブジェクトを動かします。スピードは「300」にしました。




```
set the moveSpeed to 300
move graphic 1 to the points of graphic 2
```

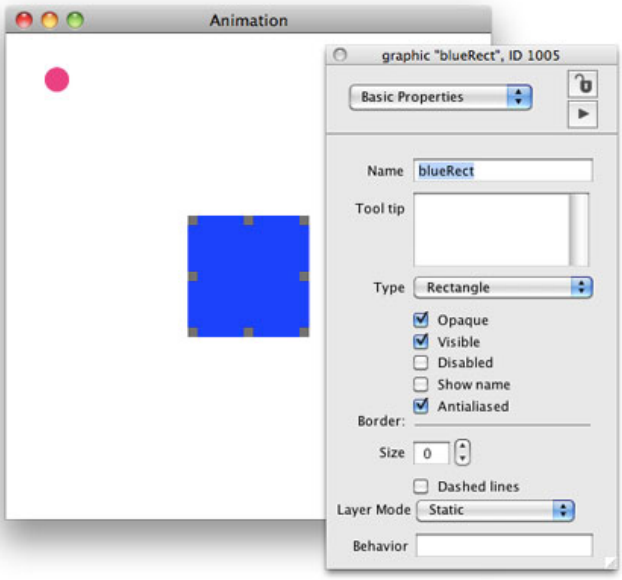
ムーブスピードを300に設定
グラフィック１(grc 1）をグラフィック２(grc 2）のポインツ（points）に、動かしなさい。

たぶん実際にはフリーハンド・ポリゴンを隠して動かす事が多いと思います。

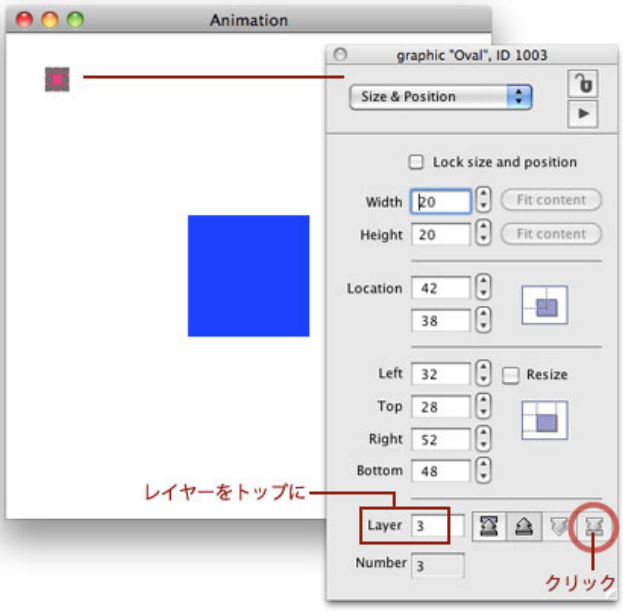
```
hide graphic 2
set the moveSpeed to 300
move graphic 1 to the points of graphic 2
show graphic 2
```

グラブ「grab」ウイズイン「within」
オーディオ・クリップ「audioClip」

カード上のオーバル（oval）を残して、フリーハンド・ポリゴン（Freehand Polygon）はデリート（delete）するか、ハイド（hide）で見えなくするかしてください（またはインスペクターで「visible」のチェックを外します）。赤い丸のオブジェクト（オーバル　oval）をマウスで移動させて、他のオブジェクトの範囲に入った時の幾つかの動作をやってみます。



ツール・パレットからグラフィックの「Rectangle」を選んで、カードのほぼ中央に100 x 100程度のブルーのバックグラウンド・カラー（backgroundColor）で名前（Name）は「blueRect」とした、グラフィック・オブジェクトを１つ作ります（上図）。最後に作ったオブジェクト「blueRect」はレイヤーの最上部にありますから、オーバル（oval）が重なった時見えなくならないように、オーバル（oval）をインスペクターの「size & Position」で、レイヤーのトップに持ってきます（下図）。

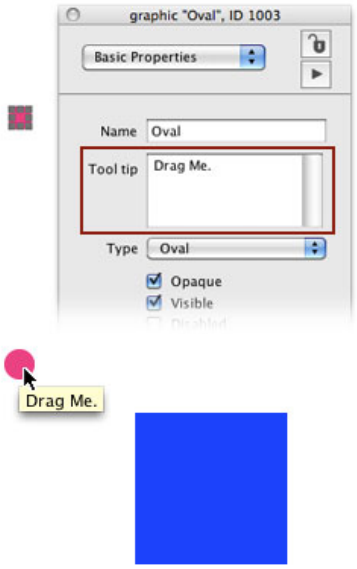


オーバル（oval）がマウスで動かせるように、オーバル（oval）のスクリプト・エディターにメッセージ・ハンドラーの「mouseDown」と、コマンドの「grab」を使ったスクリプトを書き込みます。今まではマウスが押されてマウスアップの時にスクリプトが実行される「mouseUp」を使ってきましたが、今回はマウスが押された直後のメッセージ・ハンドラー「mouseDown」です。「grab」は英語で「（何かを）ぐっとつかむ」と言った意味合いでしょうか。「me」はオブジェクトそれ自体、つまりこの場合はスクリプト・エディターのあるオーバル（graphic "Oval"）を指します。

```
on mouseDown
  grab me
end mouseDown
```

オン　マウスダウン
**　　私をぐっとつかんで（連れてって）**
エンド　マウスダウン

これで、マウスがブラウズの時（つまりランタイムの状態）に、オーバルをクリックで押さえたまま（つまりマウスダウン）でカード上のどこにでも移動できるようになりました。一般的にはドラッグ（drag）と言いますが、LiveCodeではマウスが「grab」でぎゅっとオブジェクトを掴んで、そのままマウスのロケーションをオブジェクトがフォローする動作を意味します。LiveCodeでは「drag」は別な意味で使われます（11: ２種類のイメージ参照）。




```
put the tooltip of grc "Oval"
```
グラフィック「Oval」のツールチップを返しなさい。

ついでにツール・チップ「ToolTip」について書いておきます。グラフィックやボタン等のインスペクターにある「Tool tip」に英語を書き込んでおくと、そのオブジェクトの上にブラウズ・ツールが重なった時、ツール・チップが表示されます（上図）。一般的にはマウスで移動させるのを、ドラッグと言うので「Drag Me.」としました。インスペクターの「Tool tip」には日本語は書き込めません。日本語のツール・チップはスクリプトで設定できますが、この章では扱いません。

次はオーバルをグラブ（grab）で移動させて、オーバルのロケーション（the loc of graphic "Oval"）が、グラフィック「blueRec」に入った（within）時に、ビープ音を鳴らします。ハンドラー・マウスダウン（mouseDown）のスクリプトはそのままで、別なメッセージ・ハンドラーの「mouseStillDown」を追加します。マウスダウン（mouseDown）はマウスが押された瞬間でしたが、マウス・スティル・ダウン（mouseStillDown）はマウスが押し続けられている間中です。オーバル（oval）のスクリプト・エディターに赤文字の部分を追加します。緑文字はすでに書き込まれているスクリプトです。

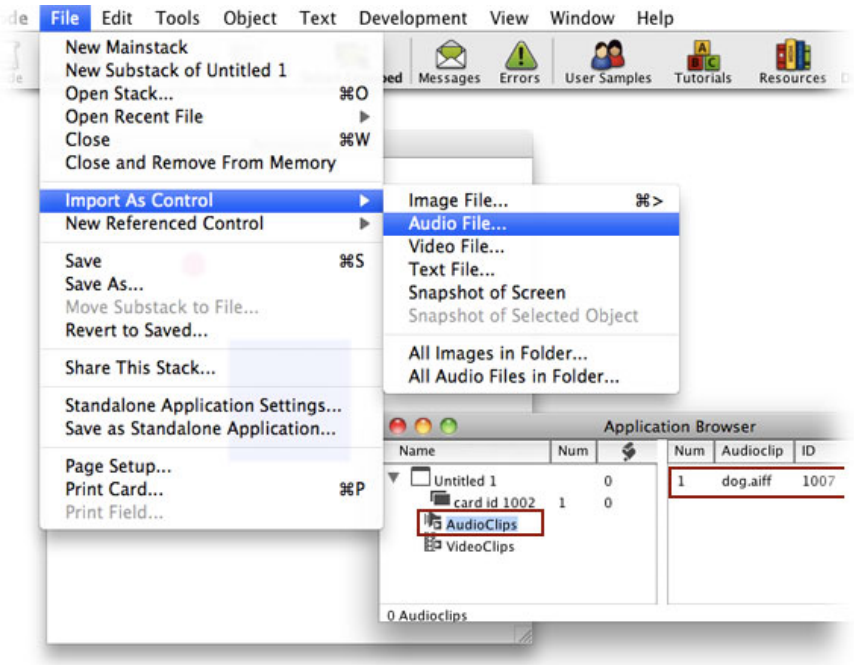
```
on mouseDown
  grab me
end mouseDown

on mouseStillDown
 if the location of me is withIn the rect of graphic "blueRect" then beep
end mouseStillDown
```

```
オン　マウス・スティル・ダウン
    もし私のロケーションが ＼
      グラフィック(grc)「blueRect」の内側だったら、ビープを鳴らしなさい。
エンド　マウス・スティル・ダウン
```

ビープ（beep）はユーザーのシステムの警告音ですから、カスタムで別なサウンドを鳴らす時には、オーディオ・クリップ（audioClip）をインポートします。インポートできるオーディオ・クリップの形式は「aiff, wav, au」のサウンドに限られます。 ダウンロードしたマテリアルから「dog.aiff」を使います。

http://kenjikojima.com/livecode/download/multiMedia13.zip

サンプルの「dog.aiff」を「Fileメニュー > Import As Control > Audio File...」からインポートします。インポートされたオーディオ・ファイルは、アプリケーション・ブラウザーの「AudioClips」の「dog.aiff」をダブルクリックすると、聞く事ができます（下図）。
注意：オーディオ・クリップ（audioClip）は、コンピュータのメディアの扱いがまだ初期の段階で作られた言葉で、カスタムのアラートや、ちょっとした効果音程度には使えますが、長い時間のファイルには適していません。MP3ファイルも使えません。MP3等の音声ファイルや、ビデオのコントロールは、プレイヤー（Player）オブジェクトを使用します。




```
play audioClip "dog.aiff"
```

```
オーディア・クリップ「dog.aiff」をプレイしなさい。
```

これでオーディオ・クリップが鳴らせます。繰り返して鳴らすのは最後に「looping」を付けま

すが、ストップする方法を用意していないと、アプリケーションを終了させない限り鳴り続けます。

```
stop playing audioClip "dog.aiff"
```
オーディア・クリップ「dog.aiff」のプレイをストップしなさい。

以上のスクリプトをまとめます。グラフィック「Oval」がマウスで移動されて、グラフィック「blueRect」に入った時ビープ音がなって、もしマウスアップされた時グラフィック「Oval」がグラフィック「blueRect」の中に残り続けたら、オーディオ・クリップ「dog.aiff」が鳴り続けます。グラフィック「Oval」がグラフィック「blueRect」から外に出た時に、オーディオ・クリップ「dog.aiff」は鳴り止みます。下の赤のスクリプトをグラフィック「Oval」のスクリプト・エディターに書き加えてください。緑文字はすでに書き込まれているスクリプトです。

```
on mouseDown
  grab me
end mouseDown

on mouseStillDown
 if the loc of me is withIn the rect of grc "blueRect" then beep
end mouseStillDown

on mouseUp
   if the loc of me is withIn the rect of grc "blueRect" then
      play audioClip "dog.aiff" looping
   else
      stop playing audioClip "dog.aiff"
   end if
end mouseUp
```

スクリプト・エディターに全て書き終わったら、試してください。

ビデオ・クリップ「videoClip」

LiveCodeのビデオ・クリップ（videoClip）も、オーディオ・クリップ（audioClip）同様にコンピュータのメディアがまだ初期の頃に作られていて、画像の位置やサイズが設定できない、ポウズ（pause）ができない等、次の章で説明するQuickTimeのプレイヤー・オブジェクトに比べると不便な点も多いのですが、短いビデオを扱う等のプロジェクト次第で使い道もあるかもしれません。プレイヤー・オブジェクトは外部のビデオ・ファイルをリンクして映像を見せますが、ビデオ・クリップはスタックに組み込むので、アプリ全体のメモリーが大きくなる事を考慮してください。

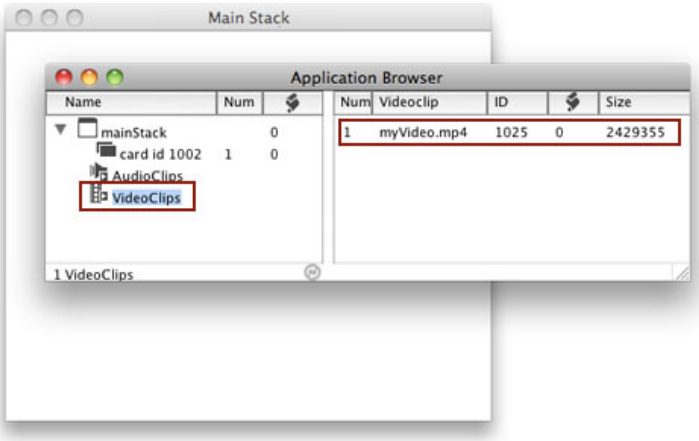
Fileメニューからメインスタックを作ります。 オーディオ・クリップ（audioClip）と同じように、ダウンロードしたマテリアルから「myVideo.mp4」を使います。
http://kenjikojima.com/livecode/download/multiMedia13.zip

やはりオーディオ・クリップ（audioClip）と同じように、「Fileメニュー > Import As Control > Video File...」からインポートします。インポートしたビデオ・クリップ（videoClip）が、アプリケーション・ブラウザーに見えます（下図）。オーディオ・クリップ（audioClip）の時のよ

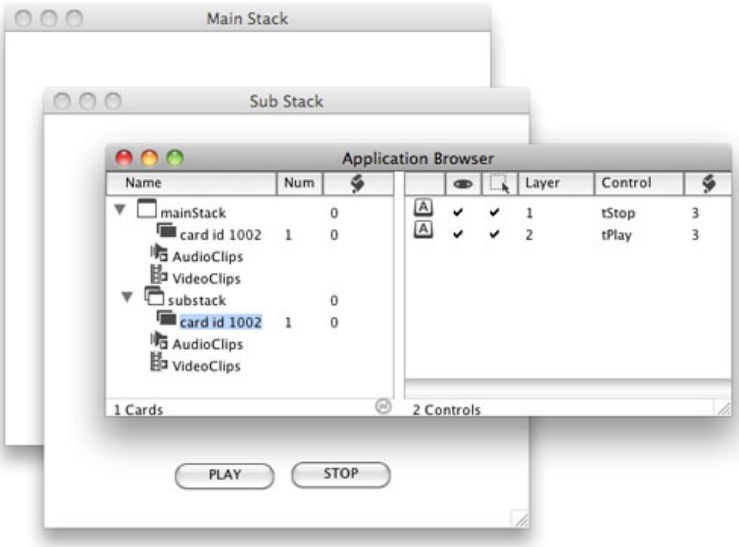
うに、アプリケーション・ブラウザー内のダブルクリックで確認しないでください。アクシデントでビデオが動き出したら、メッセージ・ボックスから

```
play stop vc
```

スクリプトの「vc」は「videoClip」の短縮形です。



このサンプルは、ビデオ用に作るサブスタックの中でビデオを映す事にします。Fileメニューから「New Substack of Untitled 1」を選んでサイズは 400 x 340 にしてください。ビデオ・クリップはカードの中央に映し出されますから、下に「PLAY」と「STOP」のボタンを置けるような大きさにしました。サブスタックの名前（name）は「substack」とします。ビデオのサイズは「376 x 200」です。サブスタックの下方に「PLAY」と「STOP」のボタンを作ります（下図）。



ビデオ・クリップはメインスタックにインポートされていますが、映し出されるのはプレイのスクリプトが実行されるカードの中央です（サブスタックのカード１）。「PLAY」と「STOP」のボタンに入れるスクリプトです。

```
on mouseUp
   play videoClip "myVideo.mp4"
```

```
end mouseUp
```

```
  オン　マウスアップ　-- これはボタン「PLAY」のスクリプトです。
     ビデオ・クリップ「myVideo.mp3」をプレイしなさい。
  エンド　マウスアップ
```

```
on mouseUp
   play stop videoClip
end mouseUp
```

```
  オン　マウスアップ　-- こちらはボタン「STOP」のスクリプトです。
     ビデオ・クリップをストップしなさい。
  エンド　マウスアップ
```

これでビデオ・クリップを操作できます。試してください。

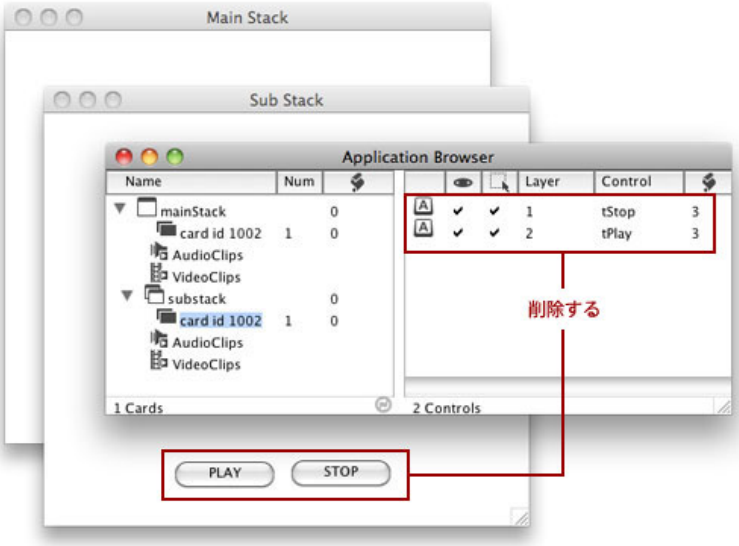
パレット「Palette」ウインドウ

もう少しプロジェクトらしさを付け加えてみます。始めにメインスタック（mainstack）の上に「ビデオを開く」ボタンを作ります。サブスタックの名前（name）は「substack」にしていますから、中に入るスクリプトは

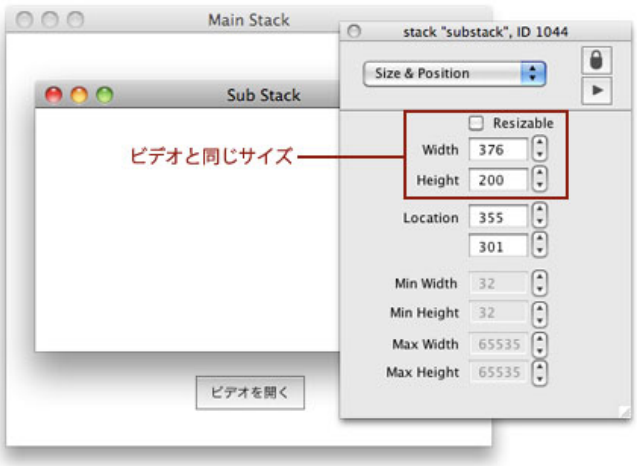
```
on mouseUp
   open stack "substack"
end mouseUp
```

```
  オン　マウスアップ
     スタック「substack」を開きなさい。　-- サブスタックのカード１が開かれます
     -- open card 1 of stack "substack"
     --　としても、同じ意味です。
  エンド　マウスアップ
```

メインスタックに作ったボタンでサブスタック「substack」が開いたら、カード上の２つのボタン「PLAY」と「STOP」を削除してください。カードのスクリプト・エディターにビデオプレイのスクリプトを書いて、サブスタック「substack」が開かれたら、すぐにビデオがプレイされるようにします（下図）。



ボタンやビデオの外の余白は必要ないので、スタック・インスペクターで、スタックのサイズをビデオと同じサイズの「376 x 200」にして、リサイザブル（Resizeable）のチェックを外します（下図）。



これでインターフェイスのだいたいの準備ができました。スタック「substack」のカード 1（card 1）のスクリプト・エディターを開いて、ボタン「PLAY」「STOP」とほぼ同じスクリプトを書き込みます。カードのスクリプト・エディターを開くのは「Objectメニュー > Card Script」を選ぶか、Mac OSなら「オプションキー + コマンドキー」、Windowsは「コントロール + オルトキー」でカード上をクリックします。カードのスクリプト・エディターに書き込む、メッセージ・ハンドラーの「openCard」はカードが開かれた時に、「closeCard」はカードが閉じられる時に、ステートメントが実行されます。以下がカード内に書き込むスクリプトです。

```
on openCard
   play videoClip "myVideo.mp4"
end openCard

on closeCard
    play stop videoClip
```

```
end closeCard
```

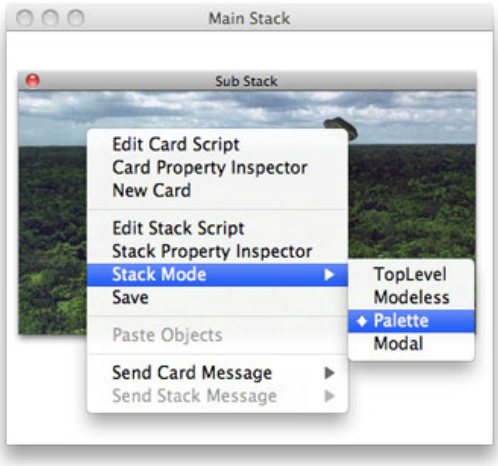
```
オン　オープンカード
    ビデオ・クリップ「myVideo.mp4」をプレイしなさい。
エンド　オープンカード

オン　クローズカード
    ビデオ・クリップをストップしなさい。
エンド　クローズカード
```

サブスタックを保存して一旦閉じます。カーソルをエディット・ツールからブラウズ・ツールに持ち替えて、メインスタックのボタン「ビデオを開く」をクリックして、サブスタックを開けてみましょう。問題なくビデオが映されたと思います。もしエラーがあったら、スクリプトを確認してください。

もうひとつスタック（ウインドウ）のスタイル（style）を修正することにします。ビデオが映し出されている間、アクシデントでユーザーがメインスタックをクリックすると、ビデオがメインスタックの後ろに隠れてしまいますから、いつでもトップにあるウインドウ・スタイルのパレット（palette）にします。現在のスタック（ウインドウ）のスタイル（style）をメッセージ・ボックスから聞いてみます。

```
put the style of stack "substack"
```
```
スタック「substack」のスタイル（style）を返しなさい。
```

「toplevel」が返されます。トップレベル（topLevel）は、普通のウインドウ・スタイルです。ウインドウのスタイルを切り替えるのは、カードからコンテキスト・メニューを出します。カード上を「マウスで右クリック」または「コントロールキー + クリック」で現れます。コンテキスト・メニューの「Stack Mode > Palette」を選んでください（下図）。



スタックのスタイルがトップレベル（topLevel）以外に変わっていると、 Mac OSでは「シフトキー + コマンドキーで右クリック」または「シフトキー + コントロール + コマンドキーでクリック」、Windows では「シフトキー + コントロールキー で右クリック」で、コンテキスト・メニューが出ます。

これで見かけ上はサブスタックがパレットになっていますが、スタックを閉じたり終了させたりした時にトップレベル（toplevel）に戻ってしまう可能性があるので、メインスタックのボタン「ビデオを開く」にパレットにするスクリプトを入れておきます。

```
on mouseUp
   palette stack "substack"
   open stack "substack"
end mouseUp
```

グラフィックの時にもスタイル（style）と言う言葉を使いました。スタイル（style）はオブジェクトの型を特定するプロパティで、スタック（ウインドウ）の型を特定する場合にも使われます。スタックのスタイルは「topLevel（普通のウインドウ）」「modeless（ダイアログ・ウインドウ）」「palette（パレット・ウインドウ）」「modal（modal以外のウインドウを編集できない）」があります。

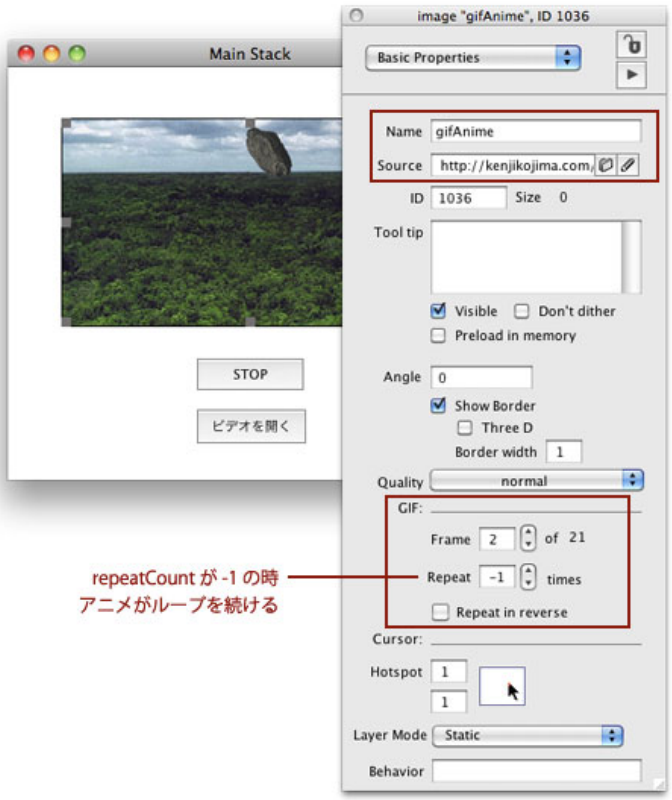
GIFアニメ

GIFアニメはイメージ・オブジェクトのファイル・ネーム（fileName）にローカルでもインターネットでもパスを入れるか、インポートでGIFイメージを直接カードに置くかの、どちらでも可能です。スタック「substack」を閉じてください。リンクでGIFアニメを見せたい時は、メインスタックの上にイメージ・オブジェクトをツール・パレットから置いて、ファイル・ネーム（fileName）を 『 http://kenjikojima.com/livecode/download/gifAnime.gif 』にセットします。ダウンロードしたサンプル・マテリアルの「gifAnime.gif」を使う時は、メニューからインポートします。イメージの名前（name）は「gifAnime」にします。GIFイメージの下に「PLAY, STOP」の為のボタンを1つ作ってください。このボタンで「PLAY, STOP」を実行して、状況によってボタンのレイベルは「PLAY GIF」または「STOP」に変えます。ボタンに入れるスクリプトは以下です。

```
on mouseUp
   if the repeatcount of image "gifAnime" is "-1" then
      set the repeatcount of image "gifAnime" to 1
      set the label of me to "PLAY GIF"
   else
      set the repeatcount of image "gifAnime" to -1
      set the label of me to "STOP"
   end if
end mouseUp
```

オン　マウスアップ

```
    もしイメージ「gifAnime」のリピートカウント（repeatCount）が「-1」だったら
        イメージ「gifAnime」のリピートカウント（repeatCount）を「1」に設定
        私のレイベルを「PLAY GIF」に設定しなさい。
    でなかったら
        イメージ「gifAnime」のリピートカウント（repeatCount）を「-1」に設定
        私のレイベルを「STOP」に設定しなさい。
    エンド　イフ
エンドマウスアップ
```



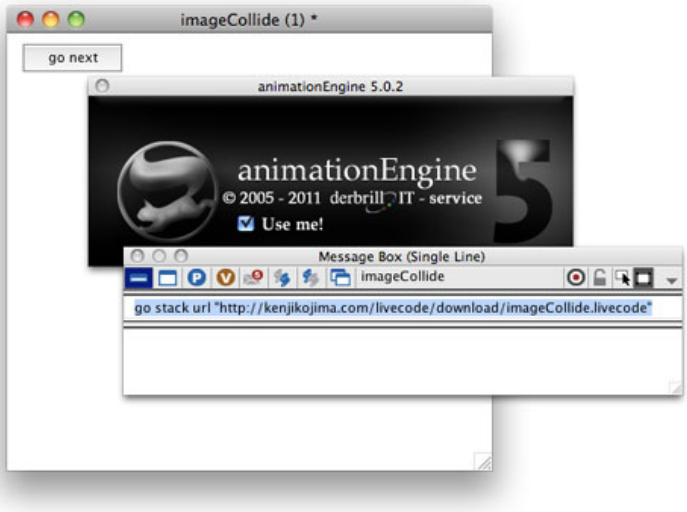
アニメーション・エンジン「animationEngine」

アニメーション・エンジンはドイツにあるサード・パーティの「derbrill社」が開発した、LiveCodeアニメーション用のコマンドやファンクションを集めた、有料のライブラリーです。LiveCodeがオープン・ソースで無料になったと同時に、「animationEngine 5.0.2」も同じライセンスのオープン・ソースになって、LiveCodeコミュニティ・エディションでも使えるようになりました。ダウンロードはこのアドレス http://forums.runrev.com/viewtopic.php?f=27&t=14399&sid=4e6165c8465c6fd1282381e35d1aa83b からできます。サイトに行ってアニメーション・エンジンのライセンスを読んで、「animationEngine5.0.2.zip」をダウンロードしてください。LiveCodeコマーシャル版に組み込む場合は、LiveCodeストアでお求めください。

ここではアニメーション・エンジンを使った 1 つのサンプル・スタックのデモを行うだけで、アニメーション・エンジンの詳しい使い方、組み込み方法等は別な処にまとめます。他のスタックが開いていたら、Fileメニューの「Close and Remove from Memory」で閉じてください。ダウンロードしたアニメーション・エンジン（animationEngine.livecode）を、Fileメニューの「Open Stack...」から開きます。

アニメーション・エンジン（animationEngine.livecode）を開いたら、「Use me!」をチェックにします。メッセージ・ボックスからスクリプトで、アニメーション・エンジンのサンプル・スタックをインターネットからLiveCodeエンジンに呼び込みます。

```
go stack url "http://kenjikojima.com/livecode/download/imageCollide.livecode"
```



「go stack url インターネット・アドレス」は、LiveCodeの書類をサーバーに置いて、LiveCodeエンジンから書類をデスクトップに呼び込んで、動作せることができます。

LiveCodeの言葉「withIn」は、レクタングル「rectangle」にポイントが入ったか、そうでないかの真偽を返すオペレイター（operater）でしたが、スタック「imageCollide.livecode」は、PINGイメージの不透明な部分が重なったかどうかを返します。ドラッグでイメージを動かしてください。

上にも書いたように、アニメーション・エンジンを使ったスタックの紹介だけで、スクリプトの解説は省きます。「imageCollide.livecode」のスクリプトは、スクリプト・エディターを開けば見る事ができます。

Tips

- フリーハンド・ポリゴン（Freehand Polygon）のポイント（points）をマウスで移動させたい時は、エディット・ツールを持ってグラフィクを右クリックでコンテキスト・メニュー（またはコントロール + クリック）を出し、「Reshape Polygon」を選びます。

- LiveCodeでは「grab」はマウスをダウンで移動させている間中、ターゲットのオブジェクトがマウスの座標をフォローします。似ている言葉「drag （11: ２種類のイメージ）」は、ペイント・ツールをある地点から別な地点に移動させるコマンドです。

- オーディオ・クリップ（audioClp）もビデオ・クリップ（videoClip）もスタックにファイルを組み込むので、あまり大きなファイル・サイズは、アプリケーション自体のメモリーを大きくしてしまうので適しません。ちょっとした効果程度に使うには簡単に扱えます。

- ゲームにはアニメーション・エンジン「animationEngine」
  http://forums.runrev.com/viewtopic.php?f=27&t=14399&sid=4e6165c8465c6fd1282381e35d1aa83b の使用を薦めます。

- スクリプト「go stack url インターネット・アドレス」を使うと、インターネットにあるスタックを、LiveCodeエンジンの動いているデスクトップに呼び出す事ができます。

この章で新しく出て来た言葉

move
コマンド（command）オブジェクトを現在のロケーションから別なロケーションに動かす
```
move graphic 1 to 20,20
```

moveSpeed
プロパティ（property）ムーブ・コマンド（move）の速度
```
set the moveSpeed to 400
```

grab
コマンド（command）マウスの動きをオブジェクトに追わせる
```
grab me
```

tooltip
プロパティ（property）ユーザーがオブジェクトをマウスで指した時に出す説明文
```
put the tooltip of grc "Oval"
```

play
コマンド（command）ムービーまたはサウンドをプレイする
stop
コマンド（command）ムービーまたはサウンドをストップする
audioClip
オブジェクト（object）サウンド・データを含むオブジェクトのタイプ　短縮形：ac
```
play audioClip "dog.aiff"
```

```
stop playing audioClip "dog.aiff"
```

mouseDown
メッセージ（message）ユーザーがマウス・ボタンを押した時に送られる
```
on mouseDown
```

mouseStillDown
メッセージ（message）ユーザーがマウス・ボタンを押している間送られる
```
on mouseStillDown
```

looping
プロパティ（property）ムービーまたはサウンドが終わったら始めからリスタートする
```
play audioClip "dog.aiff" looping
```

videoClip
オブジェクト（object）ビデオ・データを含むオブジェクトのタイプ　短縮形：vc
```
play videoClip "myVideo.mp4"
```

palette
コマンド（command）パレットのスタイルでウインドウを開く
```
palette stack "substack"
```

repeatcount
プロパティ（property）　GIFアニメが何回繰り返してプレイするか
```
set the repeatcount of image "gifAnime" to -1       -- ループし続ける
set the repeatcount of image "gifAnime" to 1     -- 1回だけプレイする
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

14: マルチメディア２

この章の概略

- *QuickTime Player オブジェクトのビデオ・ファイル設定、コントローラー（Controller）の有無、バファー（Buffer）について。*
- *ビデオのランニングタイム。LiveCodeの時間ファンクション、シェル（shell）を使ったタイムゾーンの時間の取得など少し時間の話。*
- *オープンソース版の「LiveCode Community Server」のインフォ。*
- *ボタンのアイコンをイメージにして、カスタムにします。*
- *スクロールバーとサウンドのボリューム調整。*
- *プログレスバーの作成とトラブルの対処。*

この章で使うイメージ、サンプル・スタックをダウンロードしてください。

http://kenjikojima.com/livecode/download/multiMedia14.zip

この章で使われる英単語

- buffer [動詞：（衝撃などを）和らげる]
- controller [名詞：制御・操作する器具]
- toggle [名詞：切り替えスイッチ]
- focus [名詞：焦点]
- duration [名詞：継続（時間）]
- tick [名詞：（時計の）カチカチという音, 瞬間]
- abbreviate [動詞：（語を）省略する]
- hover [動詞：（鳥、昆虫などが上を）舞う, 浮かぶ]

クイックタイム・プレイヤー

最近はWindowsでも、ほとんどQuickTimeがプレイエンストールされていますが、念のため「the qtVersion」でバージョンを調べてみます。QuickTimeがインストールされていないと「0」が返されます。

```
if the qtVersion <= 6 then answer "You need QuickTime 7 or later."
```

もしQTバージョンが６以下だったら、「QuickTime 7以上が必要です」と応えなさい。

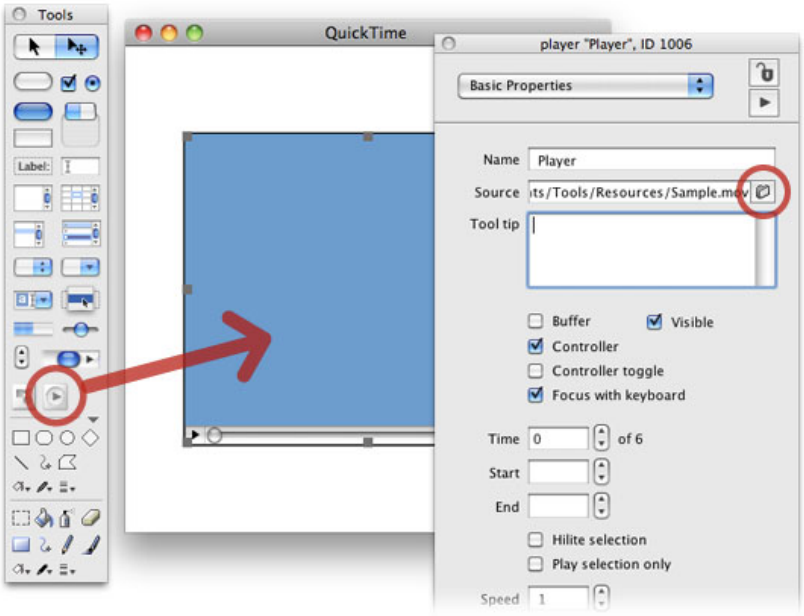
古いバージョンのQuickTimeは、フラッシュ・ファイルも再生できましたが、セキュリティ上の理由ということで、いつからでしたか再生できなくなっています。現在の私のマシーンは、ビデオ編集のためのQuickTime 7 と、QuickTime Player 10 を入れているので、「the qtVersion」では「7.6.6」が返されます。Mac OSでは「7.6.6」、Windowsは「7.7.4」が現在配布されているので、インストールされていないユーザーには「バージョン7以上が必要」と表記すれば良いかと思えます。

普通こういうチェックはスタックが開かれた直後に行われて、QuickTimeがインストールがされていなかったら機能しないアプリケーションだったら、アラートの後にプログラムを終了させます。メイン・スタックでもメイン・スタックのカード１でも、どちらのスクリプト・エディターに入れても良いです。

```
on openStack
   if the qtVersion <= 6 then
      answer "You have to install QuickTime 7 or later." &cr& \
```

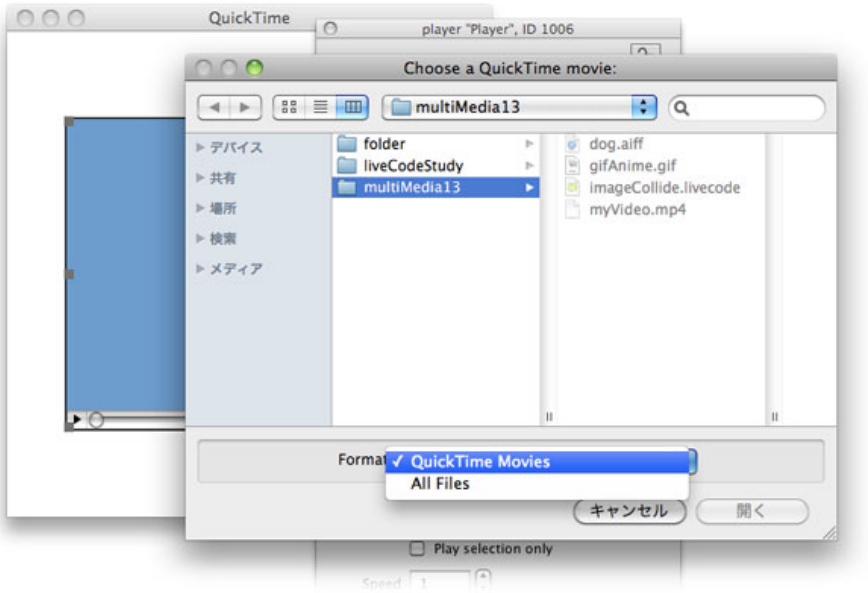
```
            "Sorry, I'm going to quit."
      quit
   end if
end openStack
```

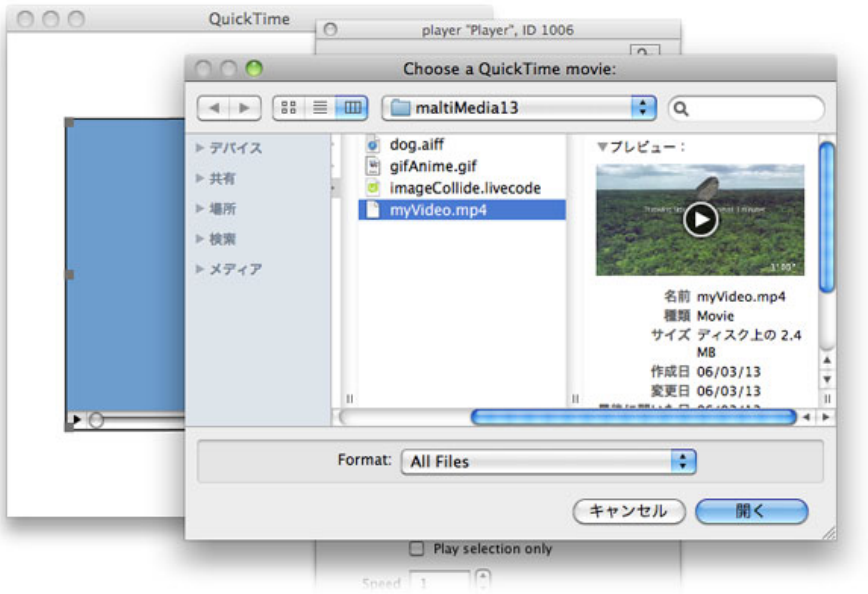
ニュー・メインスタック（New Mainstack）をFileメニューから作ってください。スタックの「name」も「title」も「QuickTime」としておきます。名前（name）は、LiveCodeでリザーブされている言葉はできるだけ避けたほうが良いですが、「QuickTime」はリザーブされていません。ツール・パレットから「QuickTime Player」をカード上にドラッグ＆ドロップします（下図）。



「Player」インスペクターの「Source（fileName）」を見ると、LiveCodeアプリに入っているリソースから「Sample.mov」がすでにリンクしてあります（上図）。「Sample.mov」は、プレイヤー・オブジェクトに見えるブルーのイメージだけの「MOVファイル」です。「QuickTime Player がサポートするメディア形式 http://support.apple.com/kb/HT3775?viewlocale=ja_JP" 」がアップルのサイトにありますから参照してください。私の場合ビデオは、オリジナルを「MOV」で作って、その後「MP4」に変換、オーディオは「MIDI」か「MP3」を使う事が多いです。「MIDI」はQuickTimeではもう公式にはサポートされていませんが、LiveCodeで再生はできます（このチュートリアルはLiveCodeコミュニティ版についてのみ書いています）。

プレイヤー・オブジェクトは、必ずファイルをリンクさせます。インスペクターで「Sample.mov」を他の「MP4」ファイルに変えてみます。「Source（fileName）」の右端にあるフォルダーのアイコン（上図）をクリックすると、ファイルをセレクトするダイアログが出ます（下図）。前章で使ったサンプル・ビデオ「myVideo.mp4」のあるフォルダーを開いても、ファイルは選べません。プレイヤー・オブジェクトはデフォルトで「QuickTime Movies（MOVフォーマット）」だけが選べる設定になっているため、「Format」を「All Files」に変更します（下図）。

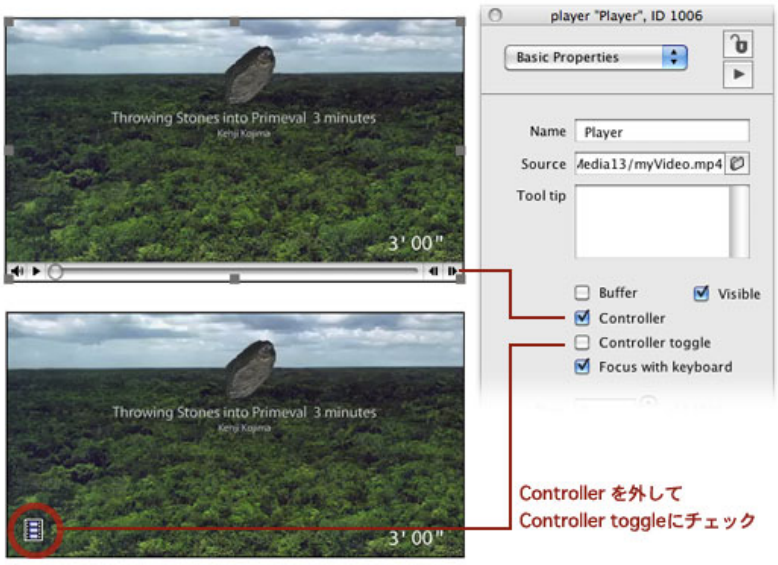




これで「myVideo.mp4」が、プレイヤーにリンクして映し出されます。これをスクリプトで書いてみます。「All Files」ではなく「MP4ファイル」のみが選べるダイアログにしました。

```
answer file "Choose a MPEG-4 Video:" with type "MPEG-4 |mp4"
if it is not empty then set the fileName of player 1 to it
```

「MPEG-4ビデオを選びなさい」とある「MPEG-4」のファイルが選べるダイアログを開きなさい。
もし「it」が空でなければ、プレイヤー1のファイルネームを「it」に設定しなさい。



プレイヤー・オブジェクトの下に見えるコントローラーで、プレイとストップと音量が操作できます。プレイヤー・オブジェクトのインスペクターを開いて見ると、上から「Buffer」「Controller」「Controller toggle」「Focus with keyboard」がセットできるようになっています。「Buffer」は後ほど説明します。デフォルトでは「Controller」がチェックされていてますが、チェックを外すと、プレイヤー・オブジェクトのコントローラーが消えます。コントローラーがない状態では、普通のイメージなのかビデオなのか、ユーザーが判断しにくいと思えたら、「Controller toggle (showBadge)」にチェックを入れると、上図のようなバッジ・アイコンがイージの左下に現れます。コントローラーをなくすとビデオ画面の天地が少し短くなります。ビデオ画面のオリジナル・サイズは「11: ２種類のイメージ」の時にも使った、「formattedWidth」と「formattedHeight」で求める事ができます。

```
put the formattedWidth of player 1 , the formattedHeight of player 1
```

イメージの時のようにサイズを変えたかったら、この２つを求めて比率で調整します。正しい画面比率でコントローラーを見せるには、天地を「formattedHeight（オリジナル）」より20ピクセル大きくしてください。

ビデオの長さ、時間の話

ビデオの長さ（時間）は「duration　デュレイション」から求めます。

```
put the duration of player 1
```

をメッセージ・ボックスから送ると「21082」が返されました。秒にするには、ムービーの１秒間のタイムスケール（timeScale）から換算します。

```
put the timescale of player 1
```

このプレイヤーのタイムスケールは「600」で返されましたから、１秒の 600 分の１と言う単位からこのビデオのデュレイション（duration）ができている事が分かります。

```
put (the duration of player 1) / 600
```

秒数「35.136667」が返されるので、35.14秒と言う事になります。

コンピュータは非常に高速で処理をするので、日常私たちが使っている時間の分や秒だけでは補えきれません。LiveCodeでは「second(s)　秒」よりも小さな単位の「tick(s)　ティクス」と「millisecond(s)　ミリセコンズ」で、それを補っています。「tick」は 1 秒の 60 分の 1 の単位です。「millisecond」は 1 秒の 1000 分の 1 の単位です。「second(s)」、「tick(s)」、「millisecond(s)」はファンクション（function）としてもキーワード（keyword）としても使われます。ファンクション（function）として

```
put the milliseconds
```

```
put the ticks
```

```
put the seconds
```

を各行メッセージ・ボックスから送ると、非常に大きな数字を返してきます。これはどちらもグリニッチ標準時（Greenwich Mean Time）1970年 1 月 1 日の午前0時0分0秒からの、ミリセコンズ（milliseconds）とティックス（ticks）、セコンズ（seconds）を返した数字で、ユニックス・タイム（Unix time）と呼ばれる時間です。

ついでなので、現在の日にちや時間のファンクションを書いておきます。

```
the date
```
「6/7/13」を返します。

```
the long date
```
「Friday, June 7, 2013」を返します。

```
the abbreviated date
```
「Fri, Jun 7, 2013」を返します。

```
the time
```
「4:03 PM」を返します。

```
the long time
```
「4:03:21 PM」を返します。

```
the abbreviated time
```
「4:03 PM」を返します。

```
put the internet date
```
「Fri, 7 Jun 2013 16:03:21 -0400」を返します。
曜日、日付、時刻に続いて最後に、グリニッチ標準時との時差

インターネットに関わって来ると、世界各地のローカル・タイムを知る必要がいろいろな場面で出てきます。LiveCodeの言葉「the internet date」で自分のいる時間帯を知る事もできますが、Unixシステムで使われるコマンドラインを、LiveCodeのシェル（shell）と言うファンク

ションを使って、世界各地のローカル・タイムを知る事ができます。シェル（shell）のこのコマンドラインは、Mac OSのターミナルで使われるものと同じで、Windowsの環境では使えません。WindowsのLiveCodeのシェル（shell）は、DOSコマンドが使えます。もしクロスプラットフォームで開発していたら、プラットフォームが「Windows」でなければ、と言う「if文」を始めに使います。

```
-- もしプラットフォームがWindowsでなければ
if the platform is not "Windows" then
      put "Asia/Tokyo" into $TZ     -- 都市名を$TZに入れる
      put shell("date") into JSTime
      put "America/New_York" into $TZ     -- 都市名を$TZに入れる
      put shell("date") into EDTime
      put "Japan Standard Time: " & JSTime & \
                  "Eastern Daylight Time: " & EDTime
end if
```

```
日本の標準時（JST = Japan Standard Time）
  「Sat Jun  8 19:37:27 JST 2013」
アメリカ東海岸時間（EDT = Eastern Daylight Time）
  「Sat Jun  8 06:40:02 EDT 2013」が返されます。
```

使われているドル・サイン「$」はUnixシステムの環境を指し示すキャラクターです。「$TZ」に都市名を入れて、「shellファンクション」のパラメータを「date」にすると、Unixで設定されたタイムゾーンの時間が返されます。ここでは「Asia/Tokyo」と「America/New_York」の２つだけののタイムゾーンですが、英語ウィキペディアの「List of tz database time zones http://en.wikipedia.org/wiki/List_of_tz_database_time_zones 」でタイムゾーンのリストが見ることができます。

「shell」を使ったUnixコマンドは、LiveCodeの使えるLinux、Macサーバーで特に重宝します。オープンソース版の「LiveCode Community Server」は http://downloads.livecode.com/livecode/server/ からダウンロードできます。
インストール等のインフォは http://livecode.com/developers/guides/server/

Windowsでタイムゾーンを取得するDOSコマンドを、私は知りません。Windowsでは「shell」を使うとコンソール・ウインドウズが開かれてしまうので、それを閉じる設定を始めに行います。Windowsの「shell」で「date」を取得する例文です。

```
set the hideConsoleWindows to true
put shell("date")
```

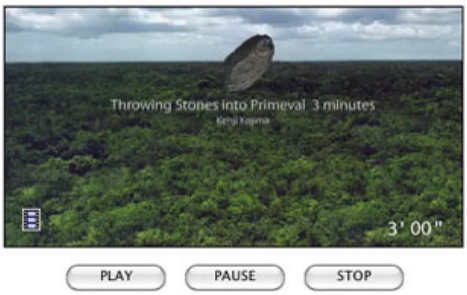
```
The current date is: 2013/06/08
Enter the new date: (yy-mm-dd)
```

の２行が返されます。

コントローラーを使わないクイックタイム・プレイヤー

コントローラー（Controller）をオフにしたプレイヤーは、インスペクターの「Foucus with

keyboard」にチェックを入れていれば、キーボードのスペース・バーでプレイ、ストップができます。しかしそれでは少し心もとないと感じた場合、「Play」「Pause」「Stop」のボタンを別に作る事ができます。ツール・パレットから「Play」「Pause」「Stop」を作ります（下図）。



名前（name）は「tPlay」レイベル（label）は「PLAY」
名前（name）は「tPause」レイベル（label）は「PAUSE」
名前（name）は「tStop」レイベル（label）は「STOP」にしました。

```
-- 「PLAYボタン」に入れるスクリプト
on mouseUp
   play player 1
end mouseUp

-- 「PAUSEボタン」に入れるスクリプト
on mouseUp
   set the paused of player 1 to true
end mouseUp

-- 「STOPボタン」に入れるスクリプト
on mouseUp
   stop player 1
end mouseUp
```

特に説明の必要もないくらにシンプルです。しかしこれだと「PAUSEボタン」と「STOPボタン」の違いが明確ではありません。プレイヤーをストップさせた時点で、ビデオを始まりか終わりに持って来ると言う方法を取ります。現在のビデオの位置（時間）は「the currentTime of player 1」で得る事ができますから、カレント・タイム（currentTime）を「0」にセットすれば、ユーザーがストップのクリックでビデオの位置（時間）は最初に設定されます。

```
--「STOPボタン」に入れるスクリプト
on mouseUp
   stop player 1
   set the currentTime of player 1 to 0
end mouseUp
```

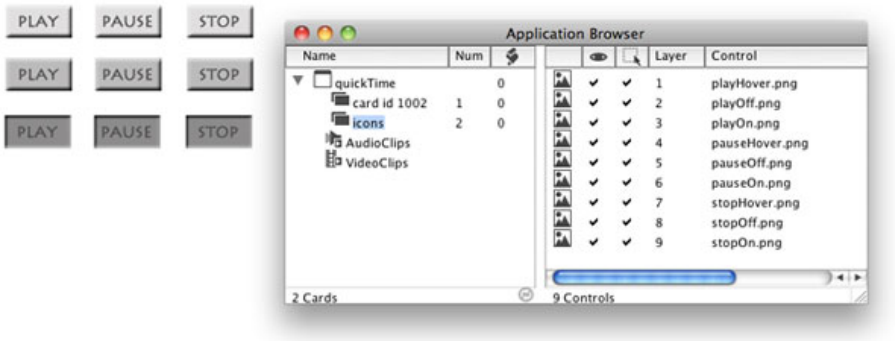
赤文字が追加するスクリプトです。

ボタンをイメージにしてカスタムに

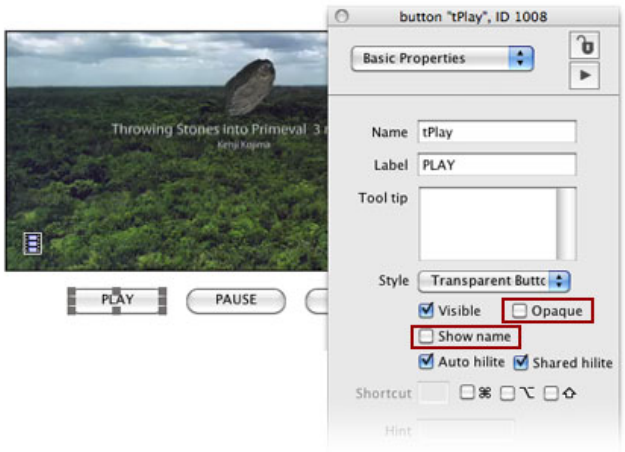
http://kenjikojima.com/livecode/download/multiMedia14.zip

ボタンにイメージのアイコンを表示させます。それぞれダウンロード・ファイルに３種類のイメージを用意しました。１）普通の状態のボタン。２）マウスが被さった時のボタン。３）クリックされた時のボタンの３種類です。今回は四角にしていますが、PINGかGIFFの透明イメージを作れば、丸にでも三角でも、好きな形ができます。（PINGの方が仕上がりはきれいです）

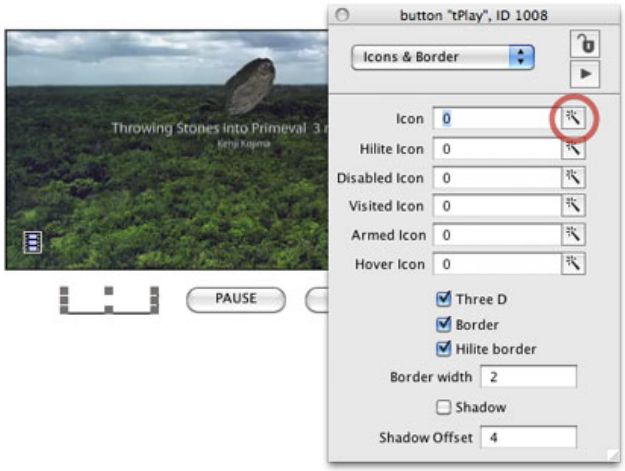
Fileメニューの「Import As Control > Image File...」からカード上にイメージをインポートすると、カードの中央にイメージが現れます。インポートしたイメージは、ユーザーが見えない場所、例えば「Xのマイナスロケーション」とかに移動させれば良いのですが、アイコン・イメージを沢山必要な場合、私はアイコン専用のカードを作ってそこに並べて置く事が多いです。必ずそうしなければいけないと言う事ではありません。



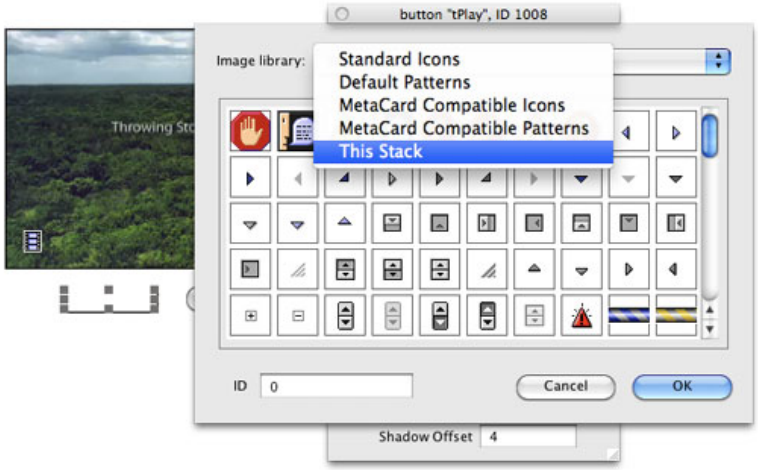
イメージをインポートしたら、目的のボタンのインスペクターで「Show name」のチェックを外します。もしイメージに透明部分がある時は「Opaque」のチェックも外して、透明なバックグラウンドのボタンにします。



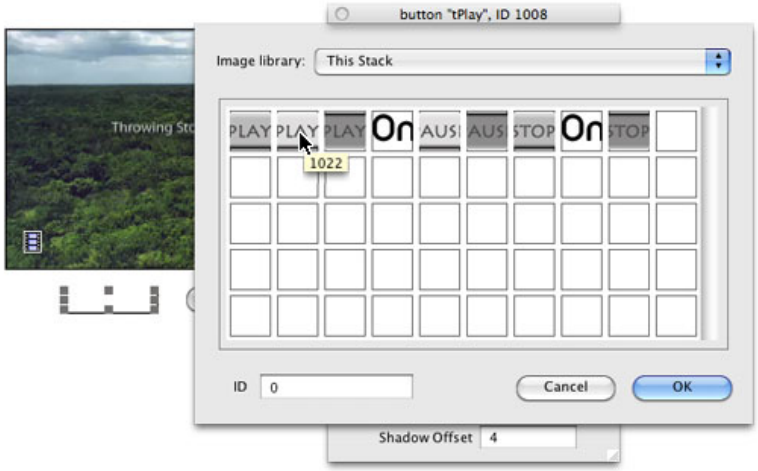
インスペクターの「Icons & Border」を開いて、「Icon」の右端のマジック・ワンドのようなアイコンをクリックすると（下図）



イメージ・ライブラリーが開きます。メニューから「This Stack」を選ぶと、このスタックにインポートしたイメージのリストが出ます。

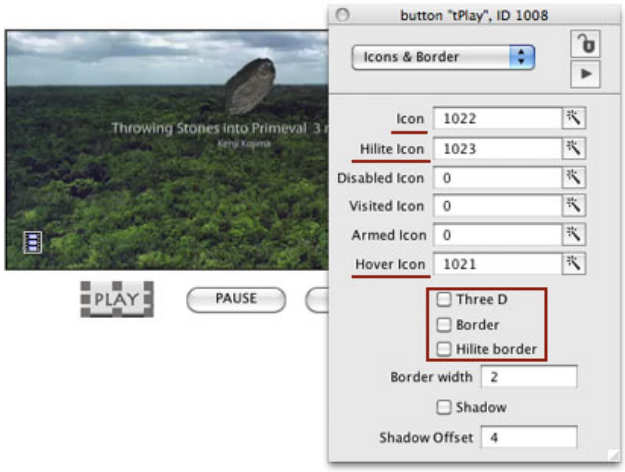


このイメージのライブラリーは、正しいイメージを表示しない事が時々あります（下図）。マウスを被せた時に出て来るツールチップは、インポートしたイメージの「ID」です。 アイコンとして表示するイメージをクリックで選んだら、「OK」で「Icon　アイコン」がセットされます。

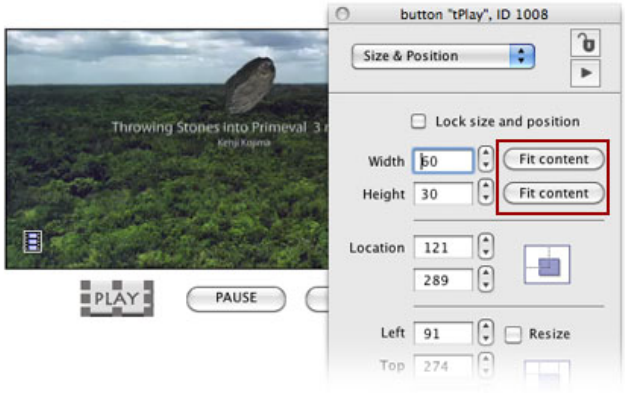


インスペクターの「Icon」に出て来る数字は、イメージの「ID」です（下図 1022）。IDが分かっていれば、イメージ・ライブラリーを開かずにIDをタイプできます。次にハイライト（Hilite Icon）のアイコンを同じ方法で、フーバー（Hover Icon）のアイコンも同じにセットします。フーバー（Hover）はマウスがターゲットの上を通過した時に反応します。イメージを表示させたボタンは、「Three D」「Border」「Hilite border」も必要がないので、チェックを外します（下図）。

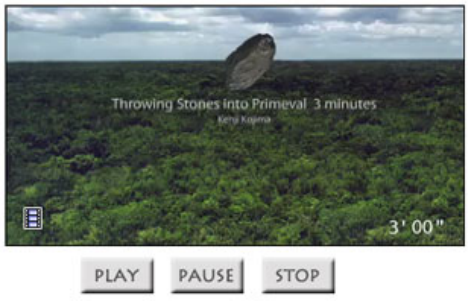
注意：IDは、イメージをスタックにインポートした時の状態によって、このチュートリアルに書いてあるナンバーとは違っている場合があります。



イメージ・サイズがピッタリとボタンのサイズに合うように「Size & Position」を開いて、Width も Height も「Fit content」をクリックして調整してください。

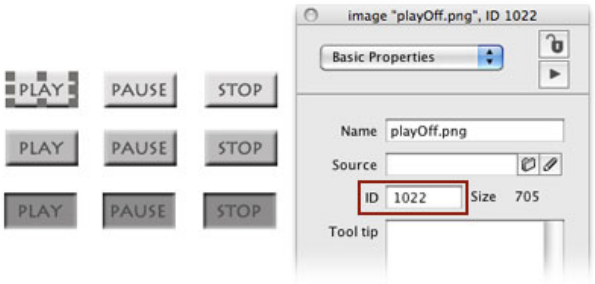


同じ作業を「PAUSE」「STOP」のボタンでも行います。



少し試してください。手順通り進めていたら、うまく行っているはずです。これで３つのボタンのそれぞれの機能は完成ですが、「PLAY」をクリックしてビデオが始まったら、「PLAY」を「PAUSE」にするという事もできるはずです。つまり「PLAY」と「PAUSE」を１つにまとめて、ボタンは２つだけに整理しても良いわけです。

手順としては、「PLAY」と「PAUSE」のスクリプトを１つにまとめて、切り替えでアイコンのイメージを変えます。アイコンのイメージは「ID」でセットされていますから、「PLAY」と「PAUSE」で使っているイメージの「ID」を調べなくてはいけません。上で見たようにボタンのインスペクターを開ければ、それぞれの「ID」は知る事ができます。また直接インポートしたイメージのインスペクターで「ID」が分かります（下図）。



「PLAY」のIDは「icon = 1022, hiliteicon = 1023, hovericon = 1021」で
「PAUSE」のIDは「icon = 1025, hiliteicon = 1026, hovericon = 1024」です。
テストに「iconTest」と言うボタンを作って、メッセージ・ボックスから送ってみます。
「button」は短縮形の「btn」を使っています。

```
set the icon of btn "iconTest" to 1025
```
「PAUSE」の普通のボタンがセットされます。

```
set the icon of btn "iconTest" to 1022
```
「PLAY」の普通のボタンがセットされます。

「PLAY」と「PAUSE」のスクリプトを１つにして、アイコンが切り替わるボタンにします。

```
-- button "tPlay" に入るスクリプト
on mouseUp
   if the icon of btn "tPlay" is 1022 then
      play player 1                          -- ビデオが動き始める
      set the icon of btn "tPlay" to 1025    -- イメージ "PAUSE"
      set the hiliteIcon of btn "tPlay" to 1026
      set the hoverIcon of btn "tPlay" to 1024
   else
      set the paused of player 1 to true     -- ビデオがポウズで止る
      set the icon of btn "tPlay" to 1022    -- イメージ "PLAY"
      set the hiliteIcon of btn "tPlay" to 1023
      set the hoverIcon of btn "tPlay" to 1021
   end if
end mouseUp
```

これで、ボタン「PAUSE (tPause)」は、必要なくなりましたから削除してください。実はもう

ひとつ修正しなくてはいけない箇所があります。ボタン「tPlay」のアイコンが「PAUSE」の時、ボタン「STOP」がクリックされる可能性があるので、「STOP」がクリックされたらボタン「tPlay」のアイコンは必ずイメージ「PLAY」に戻さなくてはいけません。

```
-- button "tStop" に入るスクリプト
on mouseUp
   stop player 1
   set the currentTime of player 1 to the endTime of player 1
   set the icon of btn "tPlay" to 1022   -- イメージ "PLAY"
   set the hiliteIcon of btn "tPlay" to 1023
   set the hoverIcon of btn "tPlay" to 1021
end mouseUp
```

カスタム・コマンドにしてカードにまとめる

さて、機能的にはこれでも良いですが、IDナンバーが違っているだけの、同じようなスクリプトが幾つかダブっていますね。これをまとめたカスタム・コマンドにして、カードのスクリプト・エディターに入れる事にしましょう。今の処はそう複雑でもないコードですが、少し入り組んで来ると、カスタム・コマンドやカスタム・ファンクションで引き出せた方が都合の良い事も多いですし、読み返してもすっきりとします。スクリプトはまとめる習慣をつけておきましょう。カスタム・コマンドが何かよく覚えていなかったら「09: 算数クイズ 2」を読み返してください。

カスタム・コマンドは「iconSet」と言う名前にしました。パラメータは４つあります。上のスクリプトを見ると「btn "tPlay"」が幾つもダブっています。全部「PLAY」ボタンなので、カスタム・コマンドの中でも「btn "tPlay"」を使う事もできますが、あるいは後から他のボタンのアイコンも変える必要が出て来る可能性もあるので、こういう場合はできるだけ汎用で使える形にしておくのが得策です。下に書いた緑文字の１行目コマンド名「iconSet」の次にある「pBtnName」は、アイコンを変えるターゲットのボタン名です。

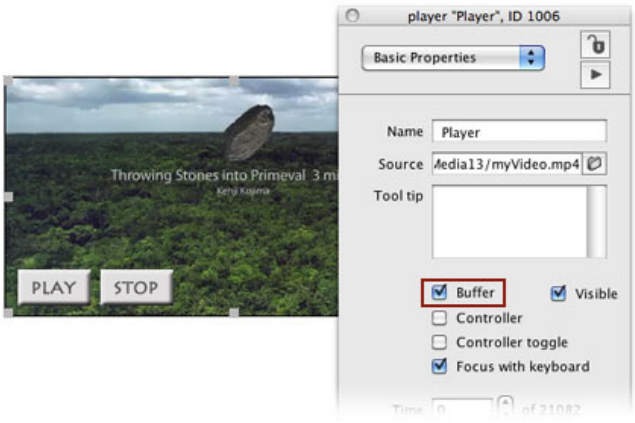
```
-- カードのスクリプト・エディターに入れるカスタム・コマンド
command iconSet pBtnName, pIconID, pHiliteID, pHoverID
   set the icon of btn pBtnName to pIconID
   set the hiliteIcon of btn pBtnName to pHiliteID
   set the hoverIcon of btn pBtnName to pHoverID
end iconSet
```

次の３つのパラメータ「pIconID, pHiliteID, pHoverID」は、オリジナル・イメージのIDナンバー

です。 ボタン「tPlay」のスクリプトをカスタム・コマンド「iconSet」を使って書き換えます。

```
-- ボタン「tPlay」の書き換えたスクリプト
on mouseUp
   if the icon of btn "tPlay" is 1022 then
      play player 1
       iconSet "tPlay", 1025, 1026, 1024   -- カスタム・コマンド
   else
      set the paused of player 1 to true
       iconSet "tPlay", 1022, 1023, 1021   -- カスタム・コマンド
   end if
end mouseUp

-- ボタン「tStop」の書き換えたスクリプト
on mouseUp
   stop player 1
   set the currentTime of player 1 to the endTime of player 1
   iconSet "tPlay", 1022, 1023, 1021   -- カスタム・コマンド
end mouseUp
```

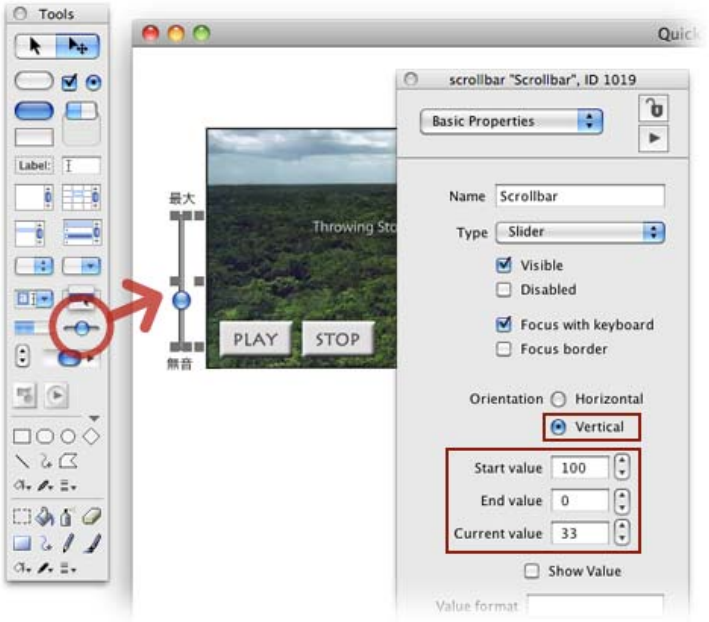
これでずいぶんスッキリしてきました。「PLAY」「STOP」ボタンもビデオ画面の中に入れられたら、レイアウト的にはスッキリする時もありますね。エディット・ツールで「PLAY」と「STOP」ボタンを、プレイヤー・オブジェクトの上に移動させてください。レイヤーでは確かにボタンの方が上にあるのに、ビデオ画面の中に隠れてしまいます。



プレイヤーのインスペクターに「Buffer　バファー」というのがあります（上図）。LiveCodeのプロパティとしての正確な言葉は「alwaysBuffer」です。「Buffer　バファー」は何かの衝撃を和らげるという意味なのですが、「alwaysBuffer」を「true」にすることで、ビデオ画面を描く為にイメージのメモリーを短時間蓄えておく処理で起きるフリッカー（ちらつき）を、和らげる働きをします。「PLAY」と「STOP」ボタンを、プレイヤー・オブジェクトの上に移動させた時に、見えなくなってしまったのは、そのフリッカーのせいだからです。インスペクターの「Buffer　バファー」にチェックを入れておけば、ボタンは見えるようになります。これはアイコンのボタンだけでなく、他のグラフィック等のオブジェクトにも有効です。
「alwaysBuffer」を「true」にすると、バッジ（Budge）は見えなくなるので、「Controller toggle（showBadge）」はtrueにしても働きません。

サウンドのボリューム調整

サウンドのボリューム調整を作ります。いろいろなボリューム調整の形体はありますが、今回はツール・パレットからスライダー（Slider）をドラッグ＆ドロップして使います。スライダーのオブジェクト名は「scrollBar」で、そのスタイル（style）を「scale」に設定した形を取っています。「scale」はインスペクターの「Type」では「Slider」となっていて、ちょっと混乱しやすいですね。オブジェクト・スクロールバー（scrollBar）はスケール（scale）の他に、スタイル（style）を「scrollbar（グループ・イメージ等をスクロールさせる）」と「progress（作業の進行状況を視覚化する）」に設定できます。スライダー（Slider）はデフォルトでは横位置なので、インスペクターの「Orientation」を「Verticle」にして、タテの設定に直します（下図）。



ボリュームのスクリプトに行く前に、スクロールバー（scrollbar）をスライドした時の数字を設定しておきます。サウンドの調整には「playLoudness」を使って「0　ゼロ」から「100」までの設定ができますから、最大を「100」最小（無音）を「0」にしますが、タテ位置のスライダーは天がスタート・バリュー（Start value）で、地（下の位置）がエンド・バリュー（End value）ですから、インスペクターに「100」と「0」を入れてください（上図）。通常のスタート・バリューの反対にしないと、扱いにくいボリューム調整になってしまいます。

Current Value(現在の値）はスクロールバー（scrollbar）をスライドさせる都度変わっていますから、設定の必要はありません。インスペクターの表示で「Current Value」としているプロパティは、LiveCodeでは「thumbPosition」と言います。英語の「thumb」は親指の事ですが、スクロールバーの丸い移動する印をサム（thumb）と言って、そのポジション（position）を「thumbPosition」と呼んでいます。「thumbPosition」は長いので、短縮形の「thumbPos（サンポス）」が、LiveCodeデベロッパーの間では普通に使われます。現在のサンポス（thumbPos）をメッセージ・ボックスから聞いてみましょう。

```
put the thumbPos of scrollBar 1
```

スクロールバー１（scrollbar 1）のサンポス（thumbPos）を返しなさい。

スタート・バリュー（Start value）を「100」にしているので、サム（thumb）をトップに持って行くと「100」を返します。ボリューム設定の言葉「playLoudness」で設定してみます。

```
set the playLoudness to 100
```
これで最大ボリュームです。

```
set the playLoudness to 0
```
これで最小ボリュームで無音です。

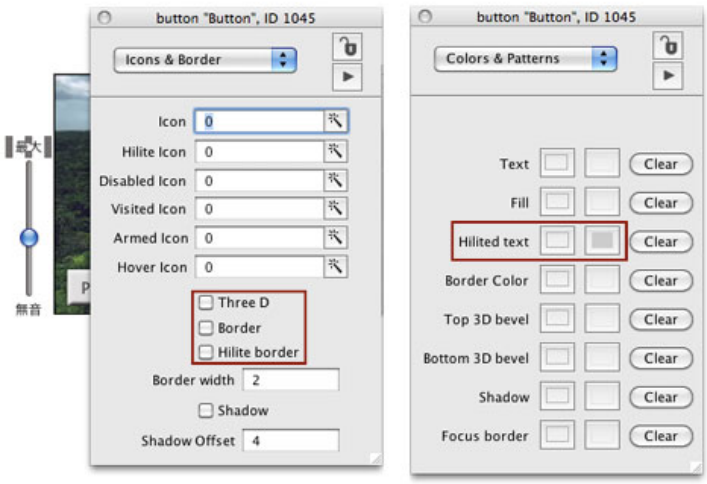
これで設定できますが、これはシステムのボリューム設定ですから、個々の「player 1」のボリューム設定にします。以下がスクロールバー１（scrollbar 1）に入れるスクロプトです。

```
on mouseUp
   set the playLoudness of player 1 to the thumbPos of me
end mouseUp
```

オン　マウスアップ
**　　プレイヤー１の音量を私のサンポス（thumbPos）に設定しなさい。**
エンド　マウスアップ

「me」と書いているのは「scrolbar 1　（sb 1）」と同じです。

スクロールバー１（scrollbar 1）のトップにスクエア・ボタン（Square Button）を置いて「最大」とレイベル（Label）に書いて、サイズはできるだけ小さく、ボトムには「無音」としたスクエア・ボタン（Square Button）を置きます。プロパティはそれぞれ好みでも良いのですが、私はボーダーを取って、ハイライトにした時のカラーをシルバーにしました（下図）。



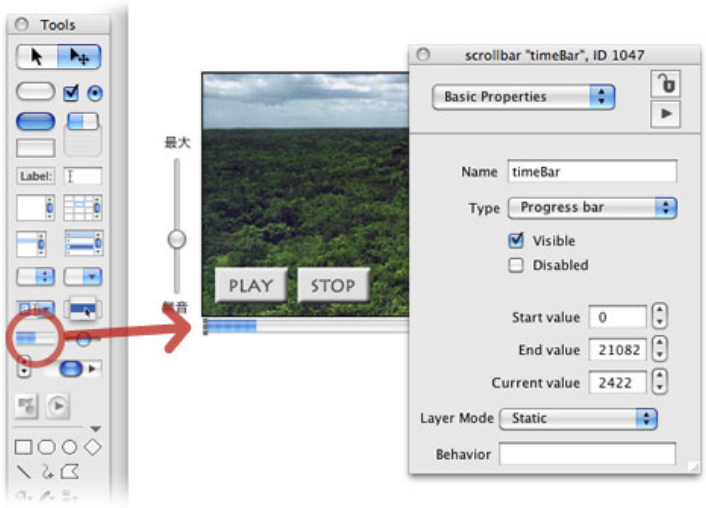

```
-- ボタン「最大」に入れるスクリプト
on mouseUp
   set the thumbPos of sb 1 to 100
   set the playLoudness of player 1 to 100
end mouseUp

-- ボタン「無音」に入れるスクリプト
on mouseUp
   set the thumbPos of sb 1 to 0
```

```
   set the playLoudness of player 1 to 0
end mouseUp
```

プログレス・バー

ダウンロードしたスタック「quickTime.livecode」を参照してください。

これから書くプログレス・バー（Progress Bar）の設定は、中級入門くらいに属する内容かもしれません。ここではビデオの時間経過を視覚化するサンプルで作っていますが、大量のデータを処理して行く経過などでも、よく見られるプログレスバーですから、考え方をだいたいでも理解しておくのが良いかと思います。



ツールパレットからプログラス・バー（Progress Bar）をプレイヤーの下に置いて、左右はプレイヤー・オブジェクトと同じ長さにします（上図）。名前（Name）は「timeBar」としました。オブジェクトのプログレス・バー（Progress Bar）は、オブジェクトのスクロールバー（scrollbar）のスタイル（style）をプログレス（progress）としたもので、設定するプロパティはスライダー（slider）とほとんど同じです。考え方としては、プログレス・バー（Progress Bar）のエンドバリュー（endValue）を、ビデオの長さ（時間 = duration）にして、ビデオのカレントタイム（currentTime）をプログレス・バーのサンポス（thumbPos）に設定して行けば良い事になります。このサンプルでは100ミリセコンド（milliseconds）毎に、バーの進行状況を書き換えて行きます。

100ミリセコンド（milliseconds）毎にバーの進行状況を書き換えると言う事は、同じステートメントを１０分の１秒毎にコンピュータに送り続けなくてはいけません。LiveCodeには「idle」と言うメッセージ・ハンドラーもありますが、こういう場合より効率が良く、負担も少ない「send コマンド」を使います。

カスタム・コマンドの名前は「videoTimer」としました。カードのスクリプト・エディターに書きます。だいたいの構成と説明（緑文字）を下に書きました。

```
1) command videoTimer     --  これはカスタム・コマンド「videoTimer」です
2)   send videoTimer to me in 100 milliseconds
```

```
3)   put the result into lVideoTimerID
4)   -- 実行されるステートメントがこの後に来る
     set the thumbpos of sb "timeBar" to the currentTime of player 1
5)   -- ストップする時の条件とその処理
     if the thumbpos of sb "timeBar" is the endValue of sb "timeBar" then
          -- コマンド「videoTimer」キャンセル
          -- プレイヤーのストップ、アイコン変更等
     end if
6) end videoTimer   --  カスタム・コマンド「videoTimer」終わり
```

1) これはカスタム・コマンド「videoTimer」です。
2)「videoTimer自身」に「videoTimer」を100ミリセコンド毎に送り（send）続けます。
3) 送った結果（the result）をIDとして「lVideoTimerID」に取得して、送り続けるのをストップする時に、そのIDをキャンセルします。「lVideoTimerID」は、他のメッセージ・ハンドラー内でも使えるように、他のオブジェクトやカード、サブ・スタックにまたがるようでしたらグローバル・コマンド、または同じオブジェクト内ならローカル・コマンドにします。上のだいたいの構成（緑文字）では、まだそれは書いていません。
4) 100ミリセコンド毎に実行されるステートメントで、プレイヤーの進行している時間（currentTime）とスクロールバー（sb）「timeBar」のサンポス（thumbPos）を同じ値にセットします。スタートする前にスクロールバーの最後の数字（endValue）を、ビデオの時間の長さ（duration) とを同じ数字に設定してあります。
5)「timeBar」のサンポス（thumbPos）のエンドバリュー（endValue）と、プレイヤーの進行している時間（currentTime）が同じになったら、IDとして取得している「lVideoTimerID」をキャンセルして、プレイヤーをストップします。その他、アイコン、ビデオ等を「PLAY」の初期設定に戻します。
6) カスタム・コマンド「videoTimer」終わり

カスタム・コマンド「videoTimer」全文（カードのスクリプト・エディターに書き込む）

```
local lVideoTimerID   -- カード内の全てのハンドラーで使えるようにローカルにする
command videoTimer
   send videoTimer to me in 100 milliseconds
   put the result into lVideoTimerID
   set the thumbpos of sb "timeBar" to the currentTime of player 1
   if the thumbpos of sb "timeBar" is the endValue of sb "timeBar" then
      cancel lVideoTimerID   -- コマンド「videoTimer」キャンセル
      set the thumbpos of sb "timeBar" to 0
      set the currentTime of player 1 to the 0
      stop player 1
      iconSet "tPlay", 1022, 1023, 1021
   end if
end videoTimer
```

カード上にあるボタン「PLAY」のスクリプトは、カスタム・コマンド「playVideo」にしてカード内に移動します。

```
-- ボタン「PLAY」に書くスクリプト
on mouseUp
  playVideo
end mouseUp
```

カード上にあるボタン「STOP」のスクリプトは、カスタム・コマンド「stopVideo」にしてカード内に移動します。

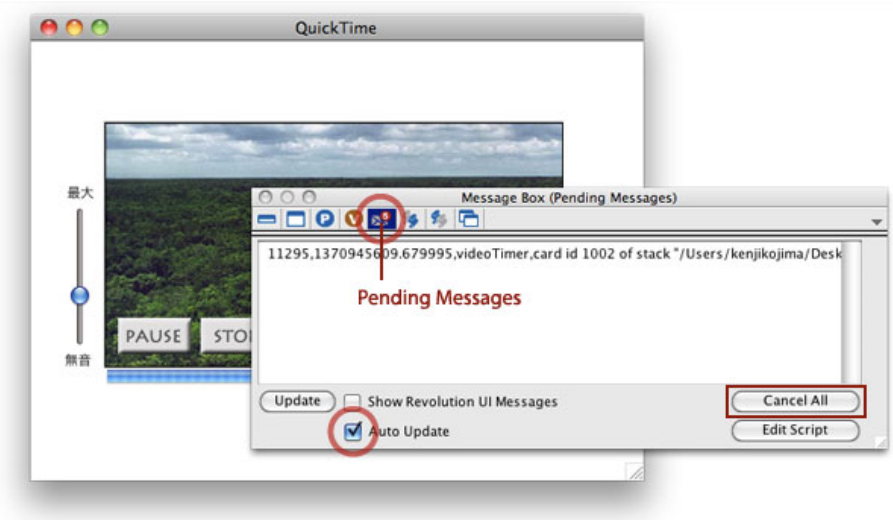
```
-- ボタン「PLAY」に書くスクリプト
on mouseUp
  stopVideo
end mouseUp
```

カスタム・コマンド「playVideo」全文（カードのスクリプト・エディターに書き込む）

```
command playVideo
   set the endValue of sb "timeBar" to the duration of player 1
   if the icon of btn "tPlay" is 1022 then
      play player 1
      iconSet "tPlay", 1025, 1026, 1024
      videoTimer    -- コマンド「videoTimer」始め
   else
      cancel lVideoTimerID   -- コマンド「videoTimer」キャンセル
      set the paused of player 1 to true
      iconSet "tPlay", 1022, 1023, 1021
   end if
end playVideo
```

カスタム・コマンド「stopVideo」全文（カードのスクリプト・エディターに書き込む）

```
command stopVideo
   cancel lVideoTimerID   -- コマンド「videoTimer」キャンセル
   stop player 1
   set the currentTime of player 1 to the 0
   iconSet "tPlay", 1022, 1023, 1021
   set the thumbpos of sb "timeBar" to 0
end stopVideo
```



「send ... to me」を使ったテストでよく起こるトラブルは、スクリプトのバグでエディターが開いて実行が途中で止っても、「sendコマンド」が送り続けられて、ストップできないと言う事があります。カスタム・コマンドを「send」した結果（the result）を入れた「ID（ここではlVideoTimerID）」を、もしグロ–バル（global）で宣言していれば、メッセージ・ボックスから

```
cancel gVideoTimerID
```

を送れば、ストップされます。しかし上のようなローカル（local）の場合は、オブジェクト内（この例ではカード内）で実行されなければ、ローカル（local）は有効ではありません。こういう場合はメッセージ・ボックスのペンディング・メッセージ（Pending Messages）を開いて（上図赤丸）、「Update」をするとペンディングされているメッセージが現れますから、「Cancel All」をクリックしてすべてキャンセルさせてストップします。

Tips

- クロスプラットフォームでQuickTimeを使うアプリは念のため、メイン・スタックを開く時にQuickTimeのバージョン・チェックで、QuickTimeがインストールされているかチェックした方が良いです。
- デフォルトの QuickTime Player オブジェクトは、インスペクターで「MOVファイル」が設定されていますが、フォーマットを変更する事で「MOVファイル」以外のビデオがセットできます。
- QuickTime Player オブジェクトのコントローラーを取り外し取り付けは、「formattedHeight」を求めてコントローラーは天地20ピクセルで調整します。
- 普通にビデオを映している状態に他のオブジェクトを乗せると、フリッカーのためオブジェクトが隠れてしまいます。プレイヤーの「alwayBuffer」をtrueに設定すると、他のオブジェクトが見えるようになります。
- 「tick」は 1 秒の 60 分の 1 、「millisecond」は 1 秒の 1000 分の 1 の単位です。
- ビデオの時間の換算は、デュレイション（the duration of player name）/タイムスケール（the timescale of player name）から求められます。
- アイコン用にインポートしたイメージは、ユーザから見えない処でしたらどこにでも置けて、アイコンとしてはIDナンバーでセットします。
- イメージ・ライブラリー（Image Library）の「This Stack」は、正しいイメージが表示できない事があります。
- イメージをセットしたとオブジェクトのサイズを一致させるのは、インスペクターの「Size & Position」で「Fit Content」をクリックします。
- 同じスクリプトが他のオブジェクトで何カ所も使われる時は、カスタム・コマンド、カスタム・ファンクションにまとめます。
- プログレスバー等の「send ... to me」を使うカスタム・コマンドが途中で止ってしまったら、メッセージ・ボックスのペンディング・メッセージ（Pending Messages）を開いて、

「Cancel All」でストップさせる。

この章で新しく出て来た言葉

qtVersion
ファンクション（function）システムにインストールされているQuickTimeのバージョン
```
the qtVersion
```

openStack
メッセージ（message）スタックが開かれた直後にカードに送られる
```
on openStack
    answer "Hi!"
end openStack
```

quit
コマンド（command）アプリケーションを終了させる
```
quit
```

showBadge
プロパティ（property）プレイヤーのバッジ・アイコンを表示
```
set the showBadge of player "myMovie" to true
```

duration
プロパティ（property）サウンドまたはムービーの時間
```
put the duration of player 1
```

timescale
プロパティ（property）サウンドまたはムービーの１秒間に繰り返す数
```
put the duration of player 1 / the timescale of player 1
```

milliseconds
キーワード（keyword）１秒の 1000 分の１
```
send videoTimer to me in 100 milliseconds
```

milliseconds
ファンクション（function）コンピュータのシステムがスタートしてからのミリセコンド
```
put the milliseconds
```

ticks
キーワード（keyword）１秒の 60 分の１
```
wait for 10 ticks
```

ticks
ファンクション（function）コンピュータのシステムがスタートしてからのティク
```
put the ticks
```

seconds
キーワード（keyword）秒の単位
```
wait for 2 seconds
```

seconds

ファンクション（function） コンピュータのシステムがスタートしてからの秒数
```
put the seconds
```

date
ファンクション（function） 今日の日付を返す
time
ファンクション（function） 現在の時間を返す
abbreviated
キーワード（keyword） date, time ファンクションの短縮形を指示する
```
put the date
put the long date
put the abbreviated date
put the internet date
```

platform
ファンクション（function） LiveCodeが走っているオペレーション・システム
```
put the platform
```

shell
ファンクション（function） UnixやDosのコマンド・ラインを実行する
```
put the platform
```

player
オブジェクト（object） ムービーやサウンドを表示する
```
play player "myMovie"
```

paused
プロパティ（property） ムービーやサウンドを一時的に停止する
```
set the paused of player 1 to true
```

currentTime
プロパティ（property） ムービーやサウンドの経過時間
```
set the currentTime of player 1 to 0
```

startTime
プロパティ（property） ムービーやサウンドの始まりの時間
```
set the startTime of player 1 to 0
```

endTime
プロパティ（property） ムービーやサウンドの終わりの時間
```
set the startTime of player 1 to 2000
```

icon
プロパティ（property） ボタンに表示するイメージ
```
set the icon of btn "tPlay" to 1025
```

hiliteIcon
プロパティ（property） ボタンがハイライトの時に表示するイメージ
```
set the hiliteIcon of btn "tPlay" to 1026
```

hoverIcon
プロパティ（property） ボタンの上をマウスが通過したの時に表示するイメージ
```
set the hoverIcon of btn "tPlay" to 1024
```

playLoudness
プロパティ（property）サウンドのボリュームを特定する

```
set the playLoudness to 100
set the playLoudness of player 1 to 40
```

thumbPosition
プロパティ（property）スクロールバーの移動できるサム（thumb）の位置　短縮形：thumbPos
scrollbar
オブジェクト（object）ポジションやセッティングを指し示す　短縮形：sb

```
set the thumbPosition of scrollbar 1 to 100
set the thumbPos of sb 1 to 0
```

local
コマンド（command）
バリアブル（variable）をローカルとして（オブジェクト内どこでも）使えるように宣言する

```
local lVideoTimerID
local lVariable1, lVariable2, lVariable3
```

global
コマンド（command）バリアブル（variable）をグローバルとして（メイン・スタック、サブ・スタック内どこでも）使えるように宣言する

```
global gVideoTimerID
global gVariable1, gVariable2, gVariable3
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

15: メニュー、タブ・パネル

この章の概略

- *LiveCodeの開発環境にあるメニュー・ビルダーを使って、基本的なメニューの構成と、スクリプトのフォーマットを作ります。*
- *メニューを設置する事で、プラットフォーム間で起きる違いについて。*
- *テキスト・フィールドで使用できる、エディット・メニューのスクリプト。*
- *ディセエイブルド、ニーモニック、ショートカット、サブ・メニュー、タブ・メニュー。条件の違いで変えるメニュー・セッティング。*
- *スタックからブラウザーでウェブページを開くスクリプト。*

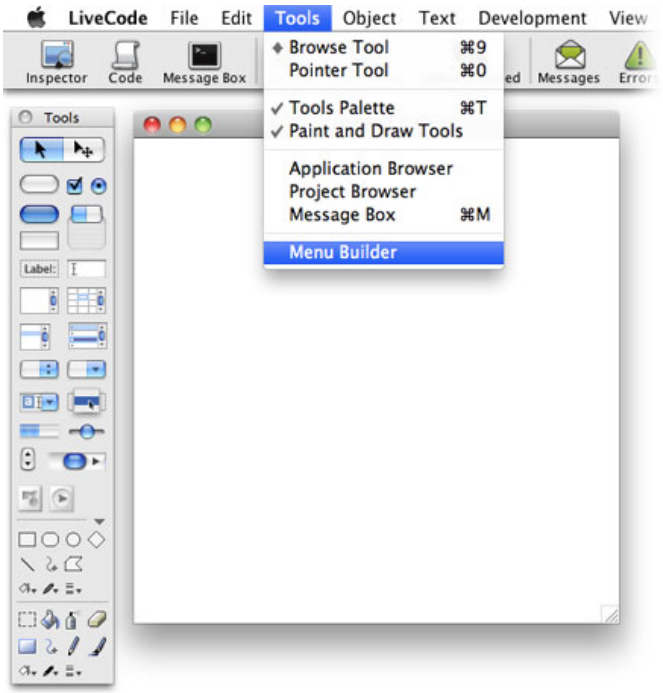
この章で使われる英単語

- Preview [名詞：下見, 試写]
- shortcut [名詞：近道]
- preference(s) [名詞：好み]
- pulldown [形容詞：縦に伸ばせる折りたたみ式の]
- platform [名詞：演台,（駅の）乗車ホーム,（コンピュータの）環境]
- mnemonic [形容詞：記憶の]
- disable [動詞：能力を奪う]
- enable [動詞：できるようにする]
- parenthesis [名詞：カッコ, パーレン]
- quotes [動詞：引用する]
- tab [名詞：つまみ, ラベル, タブ]
- panel [名詞：板, 画板, パネル]
- history [名詞：歴史, 履歴]
- acronym [名詞：頭文字を取った短縮語]

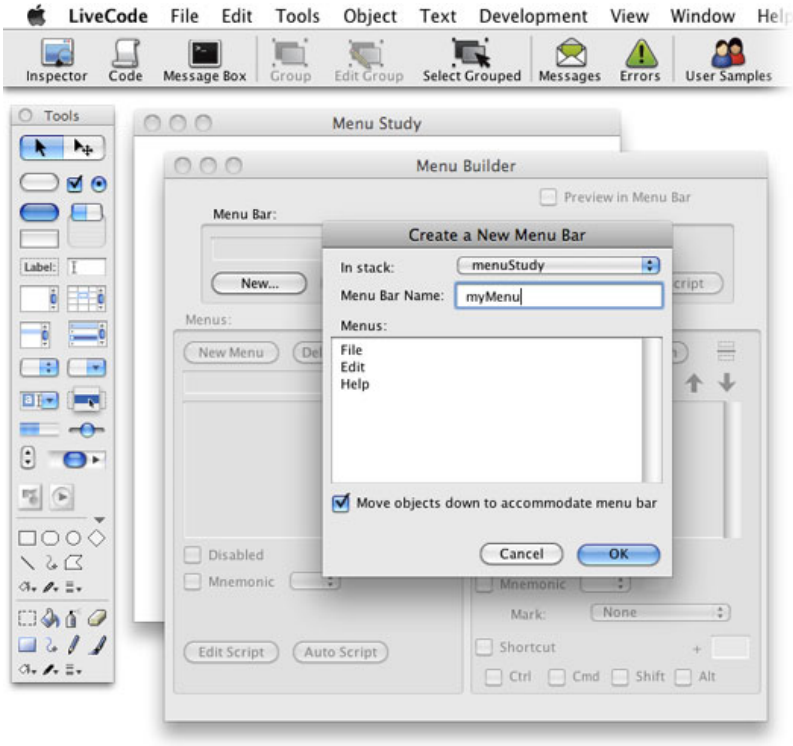
メニュー・ビルダー

いまさら書くような事でもないのですが、一般的なMac OSとWindowsののアプリケーションのメニューは、見た目が違っています。Mac OSはメニューバー（menubar）と言って、アプリケーション毎にデスクトップの最上部にメニューが並びます。それに対してWindowsでは, 個々のウインドウの内側の左最上部に小さくメニューが並びます。クロスプラットフォームで同じアプリを開発する場合は、まずその事を考慮に入れてください。

新しくメインスタックをFileメニューから作って、スタック・インスペクターでスタックの名前（name）を「menuStudy」タイトルを「Menu Study」としてください。サイズはデフォルトのままで「400 x 400」とします。「Toolsメニュー」から「Menu Builder」を開きます（下図）。

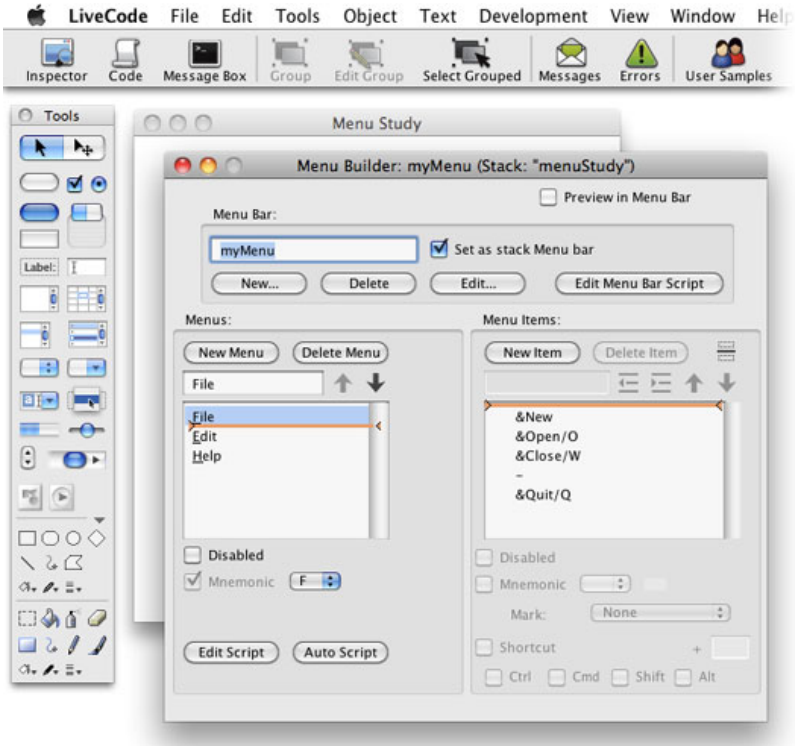


メニュー・ビルダーが開かれたら左上のボタン「New...」をクリックして 「Create a New Menu Bar」を開きます。「Menu Bar Name:」を「myMenu」とします。「Menus:」には、「File, Edit, Help」の３つのメニュー名が、デフォルトで入っています。後からでもこれらは直せますから、取り敢えずこれで「OK」をクリックします（下図）。

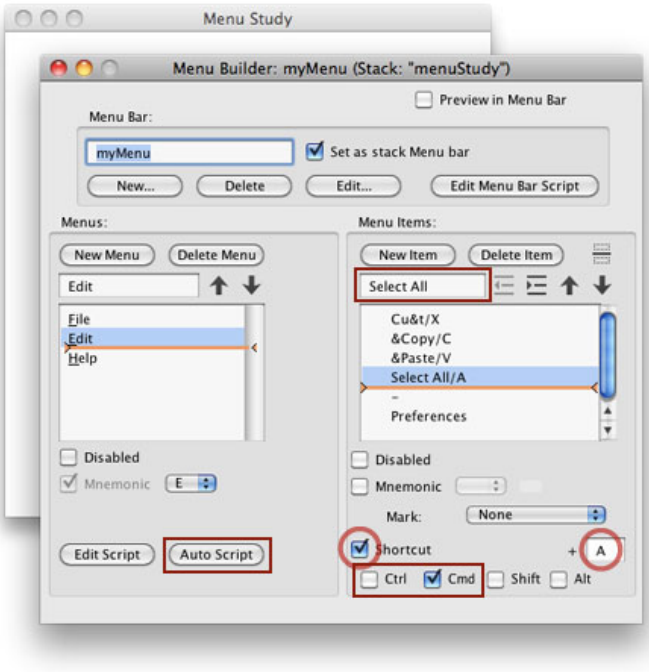


始めに開けたメニュー・ビルダーには、新しく作った「myMenu」が「Menu Bar:」に入って、左側にデフォルトの３つのメニューと、右にはそれぞれのメニューのメニュー・アイテム（menuItem）のリストが入っています（下図）。メニュー・アイテムにはちょっと見慣れない「&記号」とか斜線もありますが、後から説明します。下の図はMac OSの開発環境ですから、

メニュー・ビルダーの右上には「Preview in Menu Bar」があります。Windowsには左に「Preview ("File" Menu)」があります。Mac OSで「Preview in Menu Bar」をチェックすると、LiveCodeの編集用のメニューが見えなくなってしまうので、編集中はチェックを入れないでください。



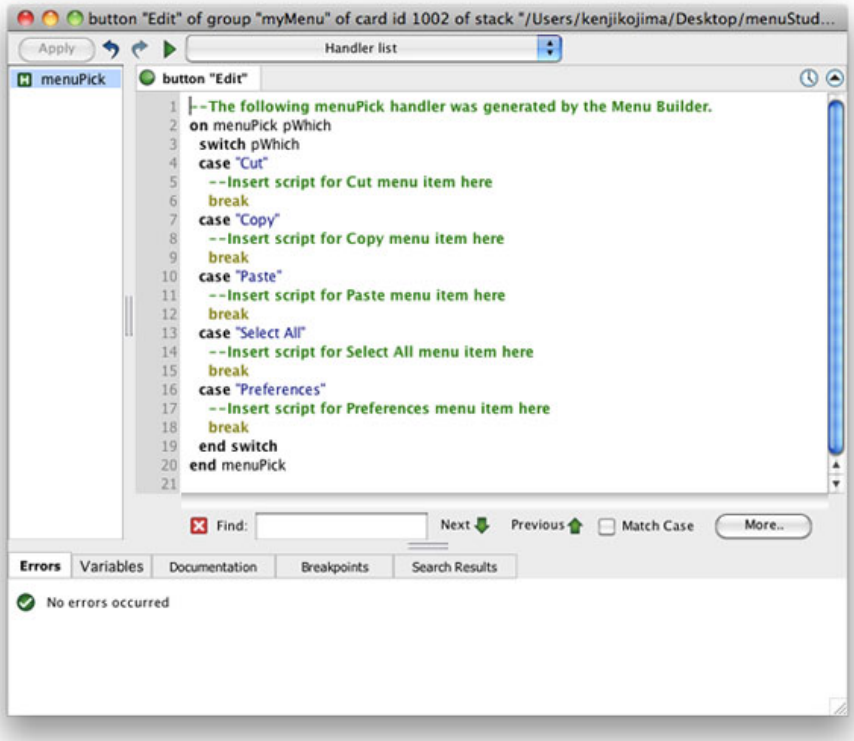
入門用には「Editメニュー」のスクリプトを書いて行くのが実用的かと思うので、左のメニューから「Edit」を選んでハイライトにしてください（下図）。右のメニュー・アイテムのリストもそれに伴って、「Editメニュー」のデフォルト・アイテムに変わります。４項目のメニュー・アイテムと、ハイフォン（-）の下にもう１項目、全部で６ラインあります。上から４番目の「Clear」を「Select All」に変更します。「Clear」をハイライトにして、上にあるアイテム名を変えられるテキスト・ボックスに、「Select All」とタイプします（下図赤枠）。



次にアイテム・リストの下の方にある「Shortcut」にチェックを入れて、その右の「＋記号」の処にショートカットのキャラクター「A」を入れます。これは「Select All」の為のショートカットです。下にある「Ctrl」と「Cmd」は、現在の開発環境がWindowsなら「Ctrl　コントロール」に、Macなら「Cmd　コマンド」にチェックを入れます（上図右下赤枠）。これはMac OSが開発環境でも、このLiveCodeドキュメントをWindowsに持って行けば、「コントロール＋A」のショートカットに自動的に切り替わりますから大丈夫です。逆にWindowsの開発環境だったら「Ctrl　コントロール」にチェックしておけば、Macになった時に「Cmd　コマンド」になります。

エディット・メニューのスクリプトを書き込む前に、左下にある「Auto Script（上図左下赤枠）」をクリックして、あらかじめフォーマットを書き込んでおくのが便利です。

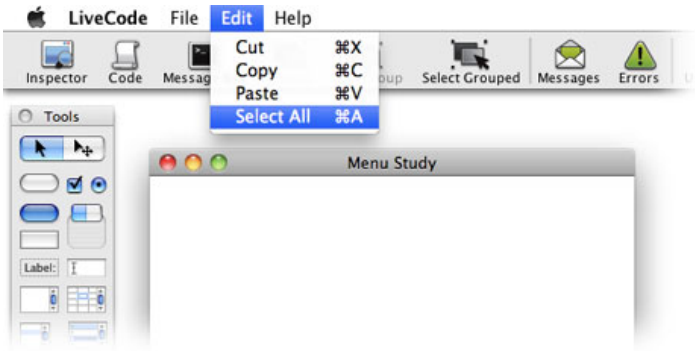
オート・スクリプトの内容のウインドウが開きます。メニュー・アイテムも「Clear」から「Select All」に直されています。これはまだ基本のフォーマットだけで、後からもう少し具体的なスクリプトを書き込んで行きます。「OK」をクリックして「Auto Script」ボタンの左の「Edit Script」をクリックすると通常のスクリプト・エディターが開かれます（下図）。



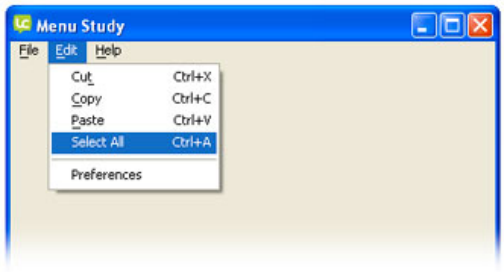
メニュー・アイテムの「Cut　カット」や「Copy　コピー」の実際のスクリプトを書き込む前に、少しメニューのオブジェクトと視覚多的な説明をします。スクリプト・エディターもメニュー・ビルダーも、ここで一旦閉じてください。

Mac OSとWindowsメニューで起こる違い

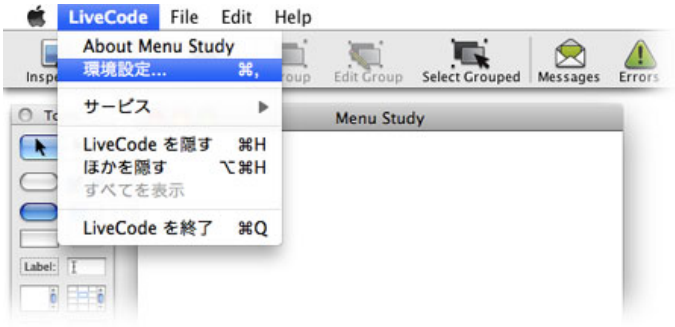
Mac OSではブラウズ・ツールを持って、開発スタック「menuStudy」をアクティブにしていれば、メニュー・ビルダーで作ったメニューの「myMenu」が、メニューバーに表示されます。「myMenu」は作ったメニュー全体に付けた名前ですから、ユーザー向けにはどこにも表示されません。「Editメニュー」をクリックしてアイテムをプルダウンさせると、４項目だけのメニュー・アイテムが見えます（下図）。



同じLiveCodeの書類をWindowsで開いて「Editメニュー」をプルダウンさせてみると、Macと同じ４項目の下にグレーのラインと「Preferences」が見えます。上にある図のメニュー・ビルダーで作った Editメニューのメニュー・アイテムを見てみると、「Select All/A」の下に「- ハイフォン」と「Preferences」があるので、Windowsではメニュー・ビルダーで見える通りですが、Macではなくなっています。



それではMacの「Preferences」はと言うと、アプリケーション名（今は「LiveCode」)のメニュー・アイテムの「環境設定」にあります。このプロジェクトが完成してスタンドアロン・アプリケーションになったら、そのプログラム名が今ある「LiveCode」に変わって、アップル・メニューの右隣に入ります。これは「メニュー・アイテムのPreferences」が、自動的に言語環境に合わせて翻訳された言葉で表示される、Mac OSのシステムに沿ったもので、同じように「Helpメニュー」にある「About」もアプリケーション名のメニューに移動します（下図）。こちらは翻訳はされずにメニュー・ビルダーで設定した「About Menu Study」で表示されます（経過は見せていませんが、「Helpメニュー」の「About」を「About Menu Study」にしました）。

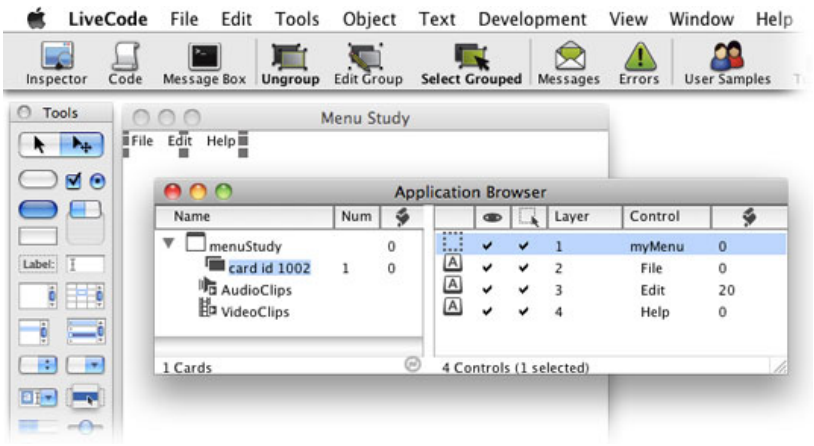


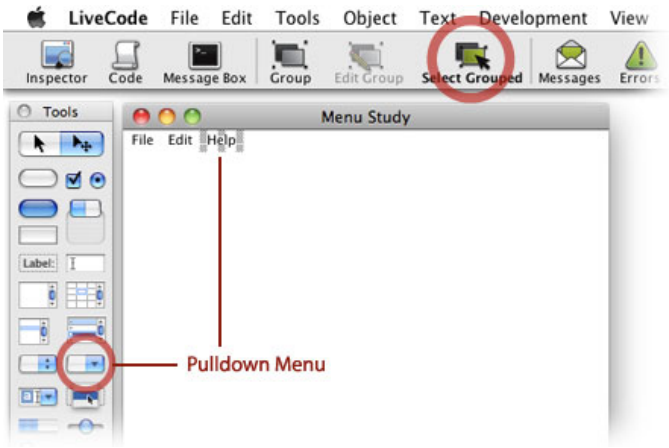
オブジェクトとしてのメニューを説明しなければいけないのだけれど、なかなかたどり着けませんね。Windowsだとメニューのオブジェクトはカードの左上に見えています。開発環境がMac OSだったら、メニューバーから取り外してみましょう。メッセージ・ボックスから

```
set the menubar of stack "menuStudy" to empty
```

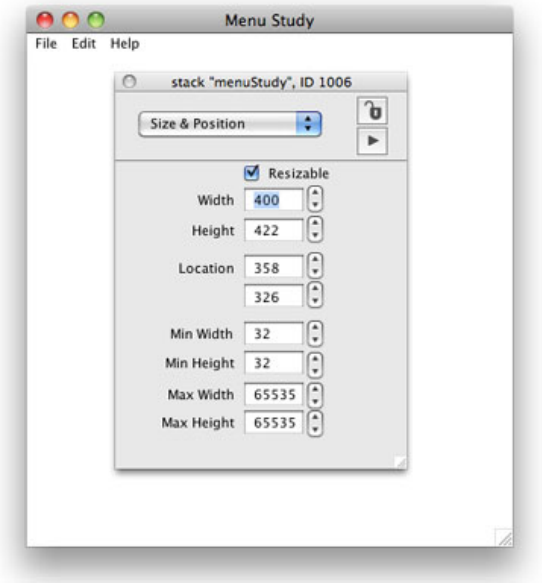
スタック「menuStudy」のメニューバーを空にしなさい。

これはメニュー・ビルダーのメニュー名の横にある、「set as stack Menu bar」のチェックを外したのと、同じ機能のスクリプトです。これでMac OSのスタックの左上にも、Windowsのメニューと同じように３つのメニューが現れます。アプリケーション・ブラウザーでカードにある「myMenu」をセレクトしてみると、３つのメニュー・ボタンをグループ化したものである事が分かります（下図）。

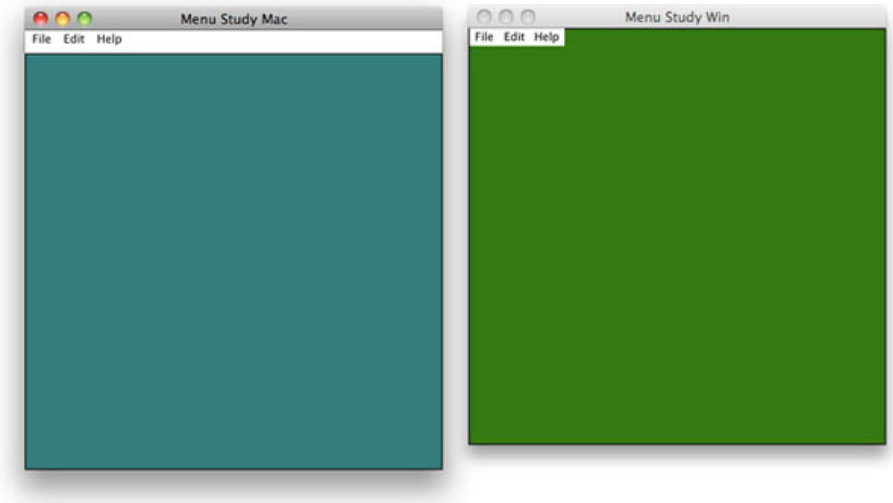




アイコン・メニューの「Select Grouped（上図赤丸）」でグループ内がセレクトできるようにすると、中のにあるボタンが選べます。これはツール・パレットから作るプルダウン・メニュー（pulldown menu）の、ボーダーを取ったボタンと同じオブジェクトです。



Mac OSが開発環境の人だったら、スクリプトでグループ「myMenu」を見えるようにした時、少しだけスタックの天地が伸びたのに気が付いたかもしれません。 開発環境がMac OSだったら、スタック・インスペクターでサイズを確認してください。「22ピクセル」デフォルトだった天地よりも伸びています（上図）。実際のアプリケーションになった時には、Mac OSではグループ「myMenu」が隠れるので、スタックの天地は400ピクセルですが、同じLiveCode書類をWindowsのアプリケーションにした場合は、メニューの分だけ大きくなって、スタックの天地が422ピクセルになる事を、心に留めておいてください。逆の場合、天地400ピクセルでWindowsで開発して、その書類でMac OSのアプリにすると、今度は天地378ピクセルのMac OSのアプリになります。

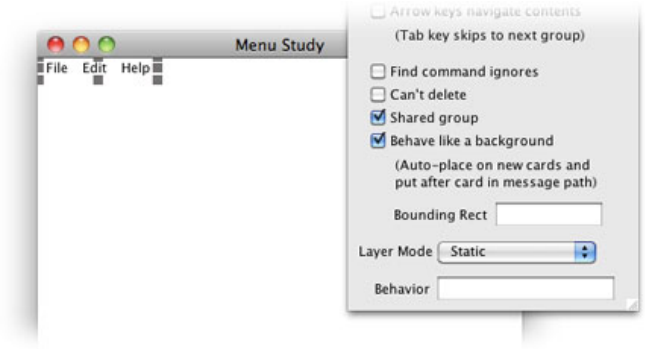


上の左の図は、開発環境Mac OSで「400 x 400」のデフォルトのメインスタックを作って、「400 x 400」のカラーのグラフィックをカードのセンターに置いてから、メニューをメニュー・ビルダーで作ったスタックです。左の図は同じように開発環境Windowsで400 x 400」のデフォルトのメインスタックを作って、「400 x 400」のカラーのグラフィックをカードのセンターに置いてから、メニューをメニュー・ビルダーで作ったスタックです。ここでは共にメニューを見せていますが、Mac OSでスタンドアロン・アプリにした場合、上のメニュー部分が隠れてスタックの天地はこれよりも短くなります。

ほとんどのアプリケーションで、この天地の違いはさほどの影響はないのですが、この事を知っていないと、どうも何か違っているなと、後から考えなくてはならなくなります。

OSの違いでメニューを設定する

グループ化したメニューのプロパティについて、もうひとつ言っておかなければ行けない事は、バックグラウンド・ビヘイビア（backgroundBehavior）にしているので、追加であたらしく作って行くカードの全てで、同じメニューとしての役割をします。



Mac OSでメニューをデスクトップのメニューバーにセットするのは、スタックのメニューバーをメニュー名（ボタンのグループ名）にセットすると言う表現します。

```
set the menubar of stack "menuStudy" to "myMenu"
```

スタック「menuStudy」のメニューバーを「myMenu」に設定しなさい。

クロスプラットフォームで同じオブジェクトをメニューに使うので、メニューの設置の違いを心配するかもしれませんが、メニュー・ビルダーの「Set as stack Menu bar」にマークしておけば、LiveCodeエンジンがプラットフォームで自動でメニューの設定をしてくれます。しかしMac OSでメニューバーにセットしていると、開発書類のスタックがアクティブになっている間は、開発書類のスタックのメニューがメニューバーに出ています。LiveCodeの編集用メニューを出すには、LiveCodeのツール・パレット等をマウスでアクティブにしなければいけません。Mac OSで開発中にメニュー・オブジェクトを出しておく事もしばしばあって、スクリプトでメニューの設置を知っておいた方が問題がありません。

クロスプラットフォームでメニュー設定を使い分けるには、スタックがオープンされる前（preOpenStack）に、プラットフォーム（platform）の違いを判断して、メニューバーにメニューを設置するか、それともスタック（ウインドウ）の上部にメニューを表示するかの指示を出します。スクリプトは、メインスタックのスクリプト・エディターに書き込みます。

```
on preOpenStack
   if the platform is "MacOS" then
      set the menubar of stack "menuStudy" to "myMenu"
   else
      set the menubar of stack "menuStudy" to ""
   end if
end preOpenStack
```

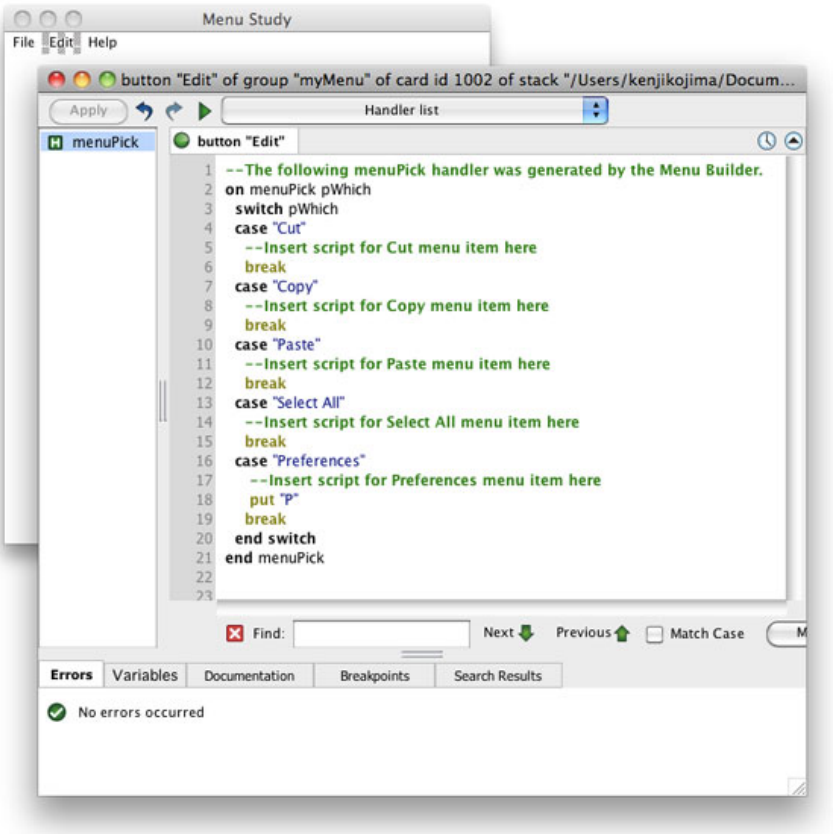
```
オン　プレオープンスタック　　-- スタックが開かれる前に働くメッセージ・ハンドラー
    もしプラットフォームが「MacOS」なら
        スタック「menuStudy」のメニューバーを「myMenu」にセットしなさい。
    そうでなかったら
        スタック「menuStudy」のメニューバーを「」にセットしなさい。　-- empty
    エンド　イフ（if文終わり）
エンド　プレオープンスタック　　-- プレオープンスタック終わり
```

エディット「Edit」・メニューのスクリプト

メニュー・ビルダーのオート・スクリプト（Auto Script）で、スクリプトのフォーマットを書き込んだままになっていますから、完成させましょう。メニュー・ビルダーをもう一度開いて「Editメニュー」のスクリプト・エディターを開く事もできますが、Mac OSだったら

```
set the menubar of stack "menuStudy" to ""
```

でメニューのグループを編集用に出して（Windowsだったらそのまま）、エディット・ツールでグループ「myMenu」をセレクトしたら、LiveCodeのアイコン・メニューの「Select Grouped」をクリックして、グループ内のオブジェクトが選べるようにします。次に「Editボタン」を選んで、スクリプト・エディターを開いてください。アプリケーション・ブラウザーでボタン・オブジェクトを選んで、スクリプト・エディターを開いても、もちろん良いです。



メッセージ・ハンドラーが「menuPick」でパラメータが「pWhich」に、「switch文」のスクリ

プトが書き込まれています。メニューピック（menuPick）は「11: ２種類のイメージ」にありましたが、「swicth文」は使わないでパラメータで写真のリンクを入れ替える時に使いました。「switch文」は「12: イメージはバイナリー・データ」で説明をしています。その時の説明を引用しましょう。

『スイッチ（switch）は、いくつかの可能性のあるバリュー（value）から、該当するものを選ぶ構文で、得られる結果は「if」とも似ています。「if」の場合は、該当しなければ、次々に「もしそうでなければ」「もしそうでなければ」と、ドンドン中に入ってチェックして行きます。「switch」は複数の可能性を並列に用意して、「case　ケース」を使って、「この場合は」「この場合は」と上から順にチェックして、該当するバリュー（value）に当たった時、その中に用意したスクリプト（ステートメント）を実行します。』

今回は「いくつかの可能性のあるバリュー（value）」は、メニューピック（menuPick）のパラメータの「pWhich」によって決まります。メニュー・アイテムのどれかが選ばれると、その値は「pWhich」に受け取られて、スイッチ（switch）の「case」に当てはまる値を上から順にチェックして、該当する値に出会った処で、そのステートメントが実行されて、「break」で他の残りのケース（case）をスキップして、スイッチ（switch）文から抜けます。

それでは実行するステートメントを完成させます。カード上にテキスト・フィールドがあって、その為のエディット（編集）機能と言う事にします。上の図のスクリプト・エディターにある２つのハイフォン（--）の付いた行にステートメントが入ります。

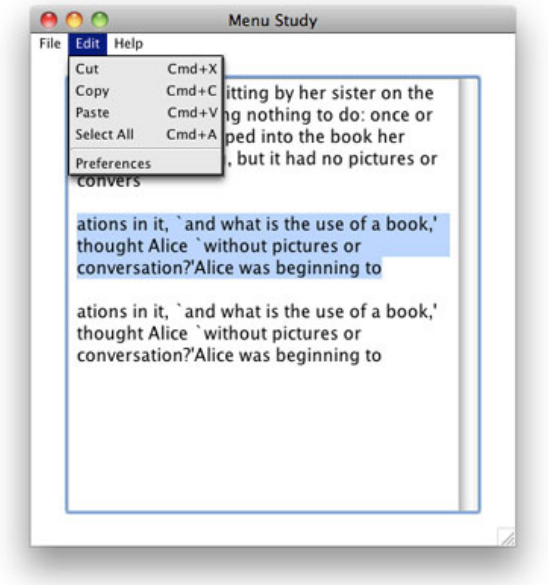
```
on menuPick pWhich
   switch pWhich
      case "Cut"
         cut
         break
      case "Copy"
         copy
         break
      case "Paste"
         paste
         break
      case "Select All"
       if the selectedField is not empty then
            select text of the selectedField
         end if
         break
      case "Preferences"
         answer "No Preferences."
         break
   end switch
end menuPick
```

```
-- この翻訳は「Select All」と「Preferences」のみを書いています。
ケース「Select All」
   もしセレクトしたフィールド（selectedField）が、空でなかったら
      そのフィールドのテキストを（すべて）セレクトしなさい。
   エンド　イフ（if文終わり）
   ブレーク　-- スイッチ文を抜ける
```

```
ケース「Preferences」
    「No Preferences.」とダイアログで応えなさい。
    ブレーク　-- スイッチ文を抜ける
```

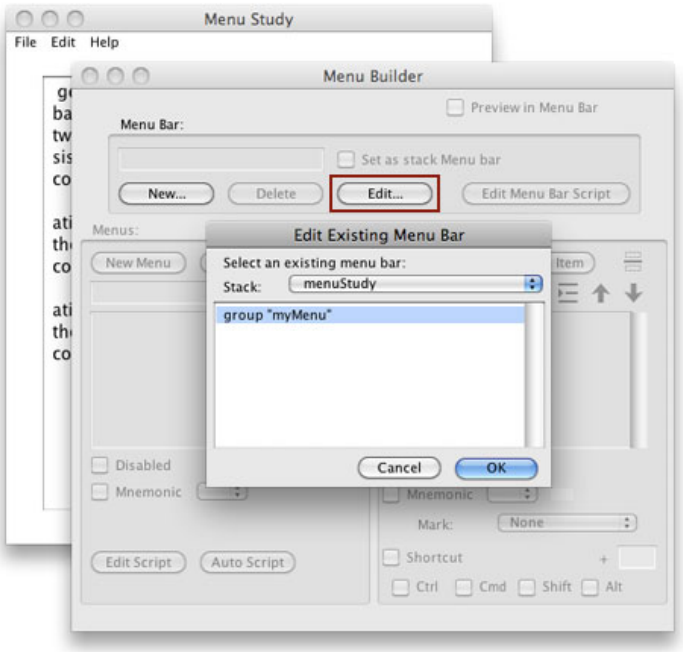
「Cut」「Copy」「Paste」は、まったくそのマンマです。「Select All」はカーソル（cursor）が入っているテキスト・フィールド（the selectedField）のテキスト全部をセレクトします。「Preferences」は、アンサー・ダイアログでメッセージを表示させています。

カード上にフィールドを作って、適当に文章を書き込んだら、エディット・メニューをテストしてください。ショートカットもうまくできますか。Mac OSだったら、メニューバーにも設定して試しましょう。

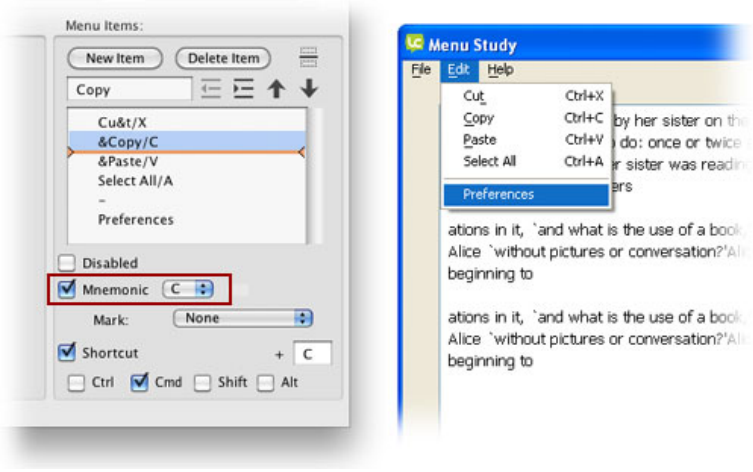


ディセエイブルド、ニーモニック、ショートカット

メニュー・ビルダーでメニュー・アイテムを見た時、「&」や「/」が見えると上に書きました。それではもう一度メニュー・ビルダーを「Toolsメニュー」から開けてください。Mac OSでスタックのメニューをメニューバーに設定していると、「myMenu」がメニューバー出ています。LiveCode開発用メニューを出すのは、LiveCodeアプリのツール等のウインドウを、カーソルでアクティブにします。

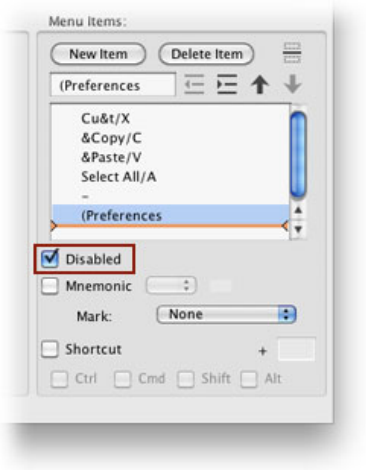


メニューをスタックに表示していると、メニューの内容が表示されないメニュー・ビルダーが出ます。「Edit（上図赤枠）」をクリックして、もうひとつウインドウを出して、どのグループをメニューにするか選びます（上図）。ここでは 1 つだけのグループですが、スタックに沢山のグループが作られている可能性があります。



メニュー・ビルダーにメニュー内容が表示されたら、左のメニュー・リストから「Edit」 を選んで、右にメニュー・アイテムを表示します。「&Copy/C」を選ぶと「mnemonic　ニーモニック」にチェックが入って「C」の文字が右に出ます（上図左赤枠）。ニーモニック（mnemonic）はMacでは使わないですが、LiveCodeのデフォルトのメニュー・アイテムのセッティングにあるので、説明をしておきます。Windows環境でオルトキー（alt key）と、メニューアイテムにアンダーラインのあるキャラクター・キーとの組み合わせて、メニューをポップダウンさせる一種のショートカットです。上の図左でハイライトされている「Copyメニュー・アイテム」で言えば、「&記号」の次のキャラクター「C」がニーモニック（mnemonic）のアンダーライン・キャラクターを示唆しています。右の図のWindowsのメニューで見てください。「Copy」では「C」にニーモニックのアンダーラインがあります。その上の「Cut」では「t」にアンダーラインがあります。上の図左でハイライトされている

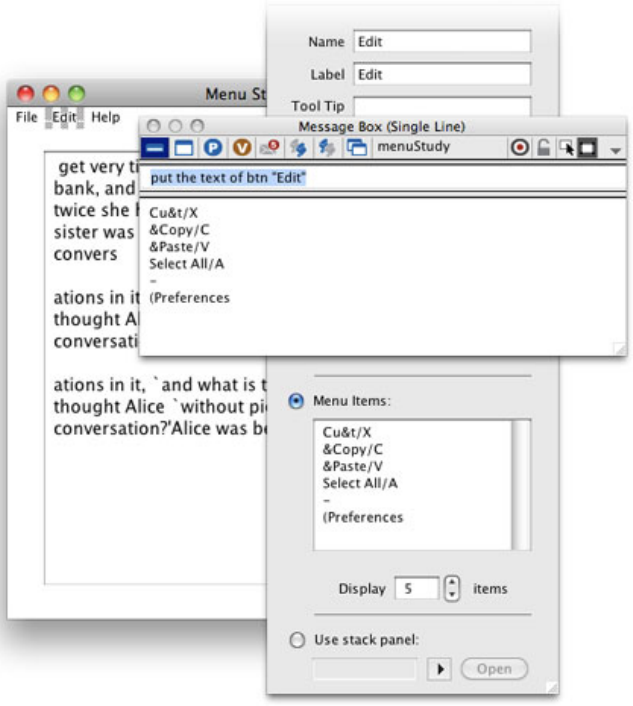
「&Copy/C」の「/　スラッシュ」の次にある「C」ば、Mac OSでは「コマンドキー＋C」Windowsでは「コントロールキー＋C」のショートカットのキャラクターを示唆しています。



メニュー・アイテムの一番下にある「Preferences」をハイライトにして選んでください。「Disabled」にチェックマークを入れると、前に左側のカッコが付いて「(Preferences」となります。これでメニュー・ビルダーを閉じるとメニュー・アイテムの「Preferences(環境設定）」は、薄いグレーになってセレクトできないディセエイブルド（disabled）になります。しかしこれはメニュー・ビルダーでやっても、必要がある時にスクリプトで「enabled　できるようにする」ができないと困りますね。メニュー・ビルダーは、大体のメニューの構成が作れれば良いくらいに考えておいて、スクリプトでも操作できるようになっていないといけません。

メニューはプルダウン・メニュー・ボタンをグループにしているので、ひとつひとつのボタンに入っているアイテムを、スクリプトで書き換えればそれが可能になります。「Edit」のボタンのアイテムを知るのは

```
the text of btn "Edit"
```



これでディセエイブルド（disabled）も、ニーモニック（mnemonic）もショートカット（shortcut）も含む、メニュー・アイテムが返されるので、「(Preferences」を「Preferences」に差し替えれば、「enabled　できるようにする」になります。

```
get the text of btn "Edit"
put "Prefrerences" into line 6 of it
set the text of btn "Edit" to it
```

ボタン「Edit」のテキストをゲットしなさい（itに入る）。
「it」のライン6に「Preferences」を入れなさい。
ボタン「Edit」のテキストを「it」に設定しなさい。

もう一度「Disabled」にするには「Preferences」の前に「(　カッコ（parenthesis）」を付けます。

```
get the text of btn "Edit"
put "(Prefrerences" into line 6 of it
set the text of btn "Edit" to it
```

「(　カッコ」はアスキー（ASCII）で「40」ですから、2行目を
put numToChar(40) & "Prefrerences" into line 6 of it
とすることもできます。

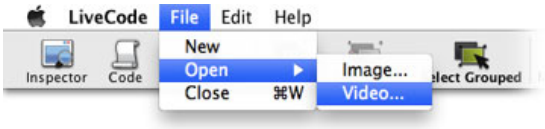
その他メニューがチェックされている印「!c」も、ワリと使われると思うので書いておきます。

```
get the text of btn "Edit"
put "!cPrefrerences" into line 6 of it
set the text of btn "Edit" to it
```

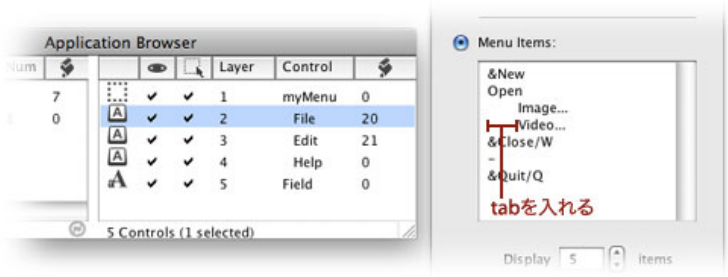
!c　-- メニューがチェックされている印

サブ・メニュー

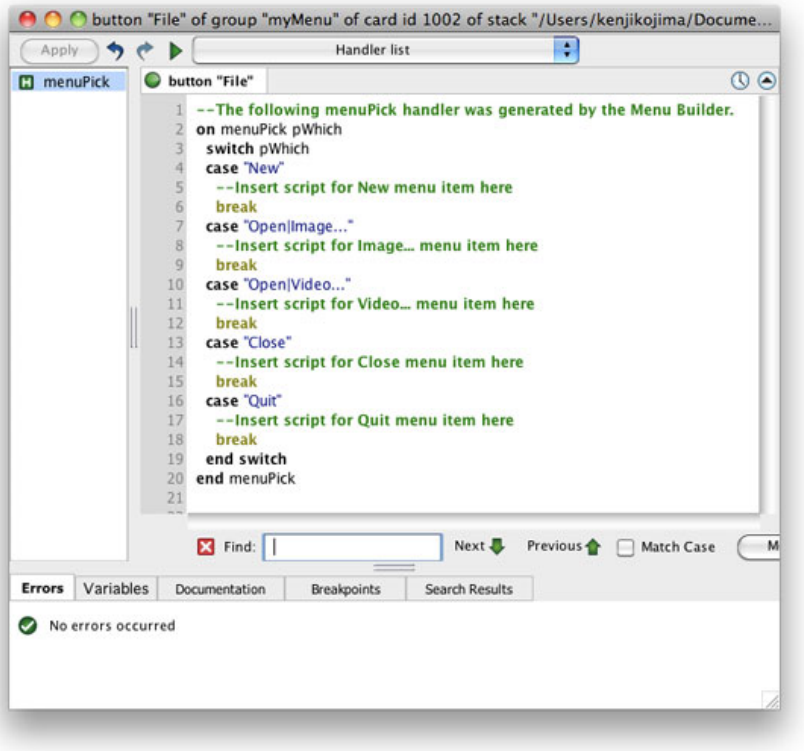
もうひとつサブメニューも重要なので書いておきます。例ではメニュー・アイテム「Open」から２つのサブメニュー「Image...」と「Video...」が引き出せるようにします。



どちらもアイテム名の後に「...」を付けているのは、そのメニュー・アイテムを選ぶとダイアログ等のウインドウが開かれると言う、ユーザー・インターフェイスの慣用的な表現で、「Preferences...」とされることも多いです（メニュー・ビルダーのデフォルトでは付いていません）。



"File"ボタンのインスペクターを開いてください。アイテム「Open」の下に２行「Image...」と「Video...」を追加します。追加した２行をサブメニューにするには、サブメニュー・アイテムの前に「tab」のアキを作ります（上図）。サブメニューの「Image...」が選ばれると、ボタンの中のメッセージ・ハンドラー「menuPick」のパラメータ「pWhich」には、タテ棒のキャラクター「l　numToChar(124)」で繋がれた「OpenlImage...」が入りますから、「switch」の「case」は「OpenlImage...」とします。



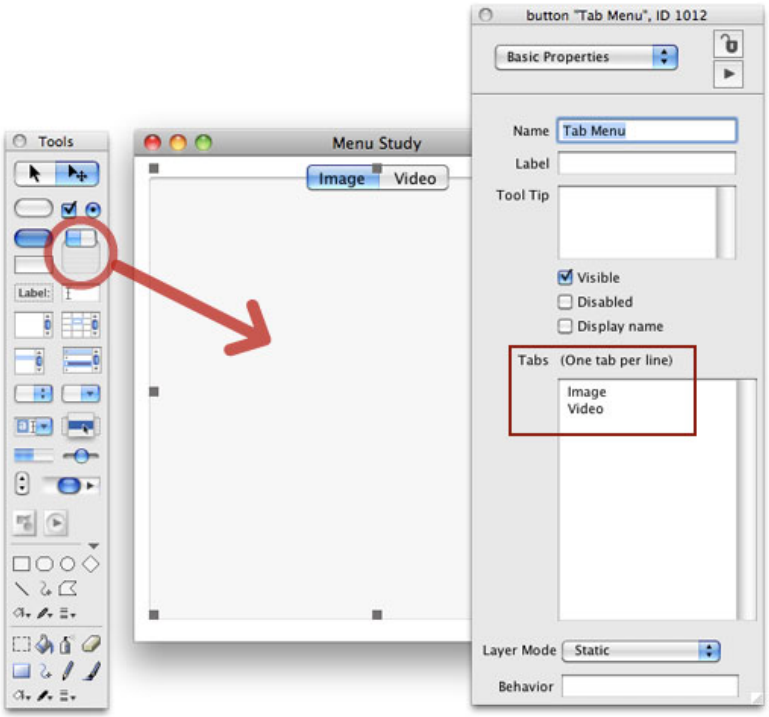

```
on menuPick pWhich
  switch pWhich
  case "New"
    --Insert script for New menu item here
    break
  case "Open|Image..."  -- サブメニュー
    answer "Selected " & quote & "Image..." & quote
    break
  case "Open|Video..."  -- サブメニュー
    answer "Selected " & quote & "Video..." & quote
    break
  case "Close"
    --Insert script for Close menu item here
    break
  case "Quit"
    --Insert script for Quit menu item here
    break
  end switch
end menuPick
```

サブメニューが選ばれると、アンサー・ダイアログを出すようにしています。ストリングスの中の文をダブル・クォートで括るのは、始めを「quote & "」で、終わりを「" & quote」にします。正確には「double quotes　二重引用符」と言いますが、LiveCodeでは「"」をquote（クオート）と書きます。

タブ・メニュー

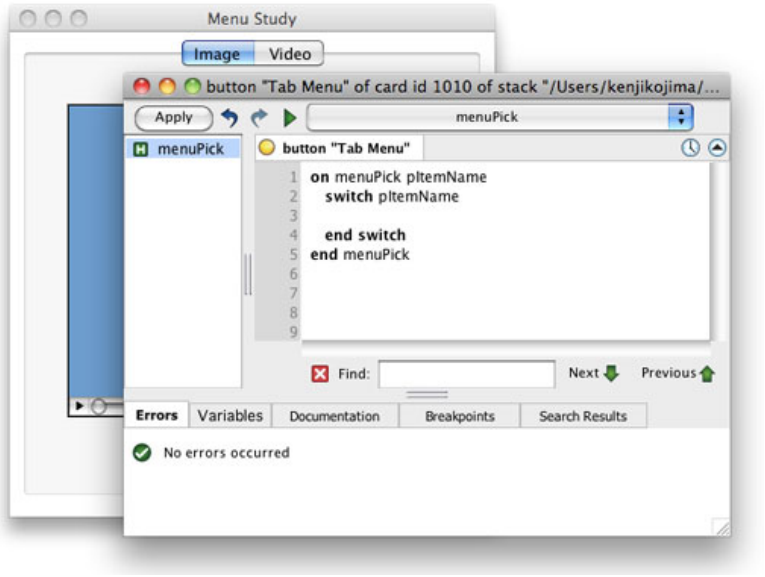
タブ・メニューを作ります。イメージとビデオをタブで切り替えるサンプルです。タブ・メ

ニューを理解するためだけのもので、実際にイメージとビデオが働くまでのスクリプトは書き込みません。上に書いたメニューから続いていて、終わりにメニューの動きディセエイブルド（disabled）と連動させます。「Obejectメニュー＞New Card」で新しくカードを作ったら、ツール・パレットから「Tab Panel」をカードにドラッグ＆ドロップします。デフォルトのタブ・パネル（Tab Panel）はタブが３つあります。Mac OSはセンターに、Windowsは左にタブが寄せてあります。インスペクターを開けると「Tab 1」「Tab 2」「Tab 3」が３行で入っています。ここでのサンプル用に「Image」「Video」と２行入れてください（下図）。タブ・パネルは２つのタブになります。



ツール・パレットからイメージ・オブジェクト（Image Area）をドラッグ＆ドロップして、300ピクセルくらいの横巾にします。その上にQuickTimeプレイヤーを乗せます。これでボタン・タブ・メニュー（button "Tab Menu"）をセレクトしてスクリプト・エディターを開けます。エディターには他のボタン・メニューと同じように「menuPick」ハンドラーと「switch」が用意されています。

```
on menuPick pItemName
   switch pItemName

   end switch
end menuPick
```



タブを切り替えた時にパラメータ「pItemName」に「Image」か「Video」が入りますから、「case」はその２つに対処します。

```
on menuPick pItemName
   switch pItemName
       case "Image"    --  「Image」タブが選ばれた
         hide player 1
         show image 1
         break
      case "Video"    --  「Video」タブが選ばれた
         show player 1
         hide image 1
         break
   end switch
end menuPick
```

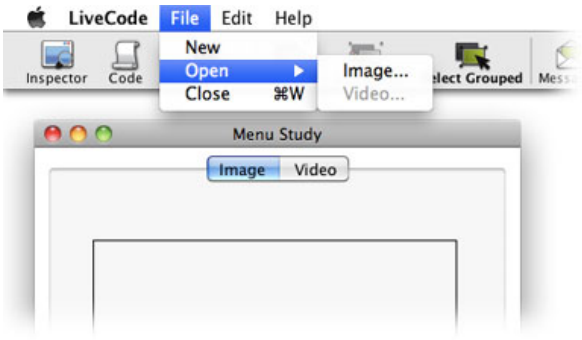
条件やカードの違いでメニューを変える

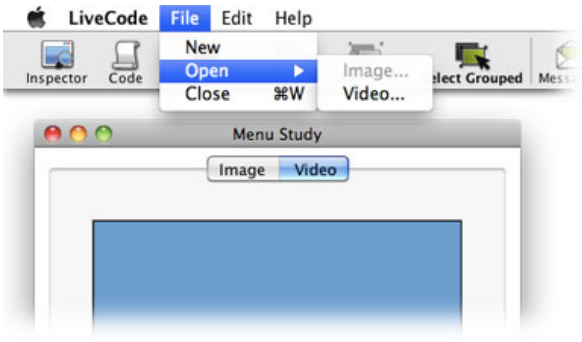
これで、切り替えができました。実際のイメージやビデオをリンクするスクリプトは、ここではやりませんが、タブが選ばれた時に「Fileメニュー」のオープンのサブメニューも同時に切り替わるようにします。

```
on menuPick pItemName
   switch pItemName
      case "Image"
         hide player 1
         show image 1
         get the text of btn "File"
         put tab & "Image..."  into line 3 of it
         put tab & "(Video..." into line 4 of it
```

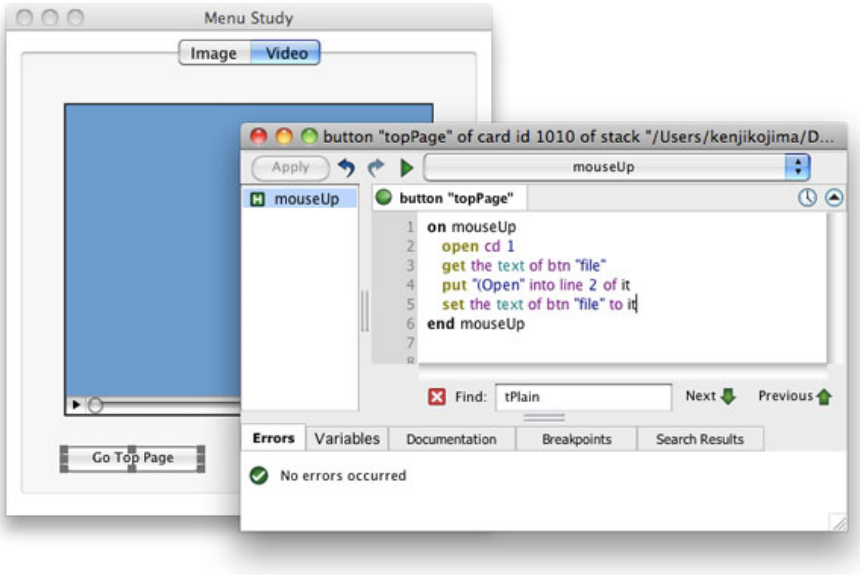
```
            set the text of btn "File" to it
            break
        case "Video"
            show player 1
            hide image 1
            get the text of btn "File"
            put tab & "(Image..."  into line 3 of it
            put tab & "Video..." into line 4 of it
            set the text of btn "File" to it
            break
    end switch
end menuPick
```

タブを切り替えると「Fileメニュー」の「Image...」「Video...」も同時に切り替わります（下図２つ）。





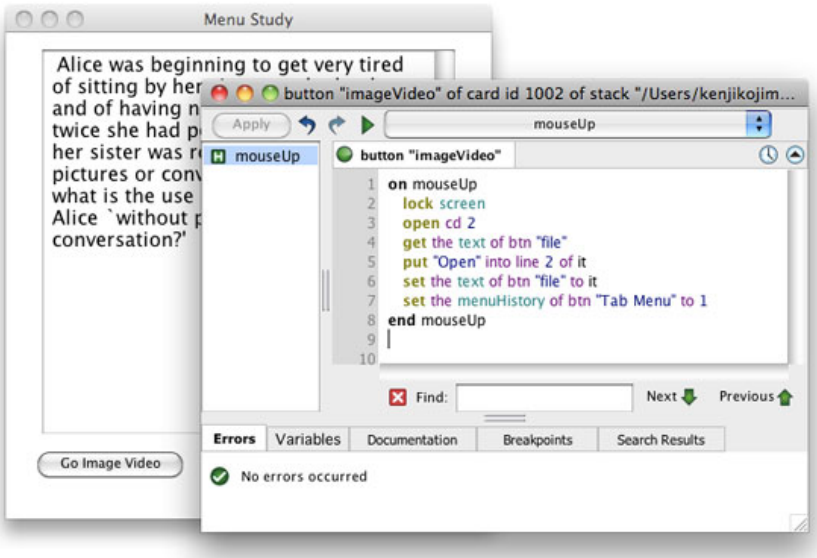
次は前のページに移るボタンを作ります。ボタンがクリックされてカード１（cd 1）が開けられると、「Fileメニュー」の「Open」を無効にして、サブメニューが開けないようにします。上でやっているスクリプトとほとんど同じです。



ボタン「Go Top Page」のスクリプト

```
on mouseUp
   open cd 1
   get the text of btn "file"
   put "(Open" into line 2 of it   -- "Open" を無効（disabled）にする
   set the text of btn "file" to it
end mouseUp
```

カード１（card 1）からカード２（card 2）を開くのもほとんど同じですが、もしカード２でビデオ・タブが開けられていたら、イメージ・タブに切り替えるスクリプトを入れておきます。



ボタン「Go Image Video」のスクリプト

```
on mouseUp
   lock screen   -- スクリーンをロックして、タブの切り替えが見えないようにする
   open cd 2   -- カード・ナンバー２を開く（スクリーンはロックされています）
```

```
   get the text of btn "file"
   put "Open" into line 2 of it   -- "Open"を有効に（enabled）にする
   set the text of btn "file" to it
   set the menuHistory of btn "Tab Menu" to 1
      -- メニュー・ボタンのメニュー・ヒストリー（menuHistory）を「1」に
end mouseUp    -- スクリーンのロックが解除されます
```

メニュー・ヒストリー（menuHistory）は、メニュー・ボタンの選ばれたアイテムを「the text」のライン・ナンバーで返します。タブ・パネルの場合は、メニュー・ヒストリー（menuHistory）をナンバーにセットすると、そのライン・ナンバーのタブをクリックしたのと同じ効果が得られます。カード２を開けてメニュー・ヒストリー（menuHistory）をセットすると、一瞬セット前のタブ・パネルが見えてしまうので、スクリーンをロックして「lock screen」、ユーザーに見えないようにします。ロック・スクリーン（lock screen）は、メッセージ・ハンドラーを抜けるとアンロック・スクリーン（unlock screen）になって解除されます。ハンドラーの中でアンロックしたい場合は、その箇所に「unlock screen」と書きます。

ウェブページをブラウザーで開ける

「Helpメニュー」に「Open Kenji Kojima's Web」を付け加える事にしましょう。スタックのメニューから、ユーザーがデフォルトに設定しているブラウザーで、ウェブページを開けるメニュー・アイテムです。「Helpメニュー」の２行目に「Open KenjiKojima's Web」を加えて、メニュー・ビルダーのオート・スクリプト（Auto Script）で、スクリプトのフォーマットを書き込みます。スクリプト・エディターを開いて

```
on menuPick pWhich
  switch pWhich
  case "Help"
    --Insert script for Help menu item here
    break
  case "Open KenjiKojima's Web"
    revGoURL "http://www.kenjikojima.com/"
    break
  case "About Menu Study..."
    --Insert script for About Menu Study menu item here
    break
  end switch
end menuPick
```

「revGoURL」はLiveCodeの以前の名前「Revolution」から来ています。LiveCodeでは始めに「rev」と付いた言葉はリザーブされているので、カスタムのコマンド、ファンクション、バリアブルでは使えません。「URL」は「Uniform Resource Locator」のアクロニム（acronym　頭文字を取った短縮語）で、一般にはウェブ・アドレスと言っているものです。

```
revGoURL "ウェブ・アドレス"
```

で、目的のページがブラウザーで開かれます。アドレスは必ずクオート（quote）で括ります。

Tips

- メニュー・ビルダーはメニューの大体の構成を作るには適していますが、開発が進むにつれて条件その他でメニューを操作しなくてはいけない箇所も出てくるので、メニューをスクリプトで操作できるようになる事が重要です。

- メニューバーを設置すると、Mac OSとWindowsではスタックの天地がメニューのサイズ分変わってくるので、クロスプラットフォームでの開発では、そのことを留意しなくてはいけません。

- メニュー・アイテムをサブ・メニューにするには、始めにタブ（tab）を挿入します。

- メニュー・ヒストリー（menuHistory）で、メニュー・ボタンをセットすると、メニュー・アイテムが選ばれたのと同じ動作になります。

- スタックからウェブページをブラウザーで開けるのは「revGoURL」を使います。

この章で新しく出て来た言葉

menubar
プロパティ（property）トップにあるスタックのメニューバーに設定されている名前
```
set the menubar of stack "menuStudy" to empty
set the menubar of stack "menuStudy" to "myMenu"
```

preOpenStack
メッセージ（message）スタックが開かれる前にカードに送られる
```
on preOpenStack
    set the menubar of stack "menuStudy" to ""
end preOpenStack
```

cut
コマンド（command）セレクトしたテキストやオブジェクトをコピーして削除する
```
cut
```

copy
コマンド（command）セレクトしたテキストやオブジェクトをクリップボードにコピーする
```
copy
```

paste
コマンド（command）クリップボードにコピーした内容を、
セレクトした処にカーソルの入っているフィールドにペーストする
```
paste
```

selectedField
ファンクション（function）テキストがセレクトされているか、
マウスが入っているフィールドのナンバーを返す
```
select text of the selectedField
```

quote
コンスタント（constant）ダブル・クオート・キャラクター（ASCII 34）
```
quote & "Image..." & quote
```

tab
コンスタント（constant）タブ・キャラクター（ASCII 9）
```
put tab & "Image..." into line 3 of it
```

revGoURL
コマンド（command）ウェブ・ブラウザーでURLを開く
```
revGoURL "http://www.kenjikojima.com/"
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

16: プロパティ、テーブル、アレイ（Array 配列）

この章の概略

- *テンプレイト（template）のオブジェクトを作り、プロパティがどう設定されているかを調べます。*
- *アレイ（array）の基礎：キー（keys)、エレメント（element）。*
- *コンバイン（combine）を使って、オブジェクトのプロパティ（properties）を調べます。*
- *コンバイン（combine）と、逆の働きをするスプリット（split）。*
- *テンプレイトのバリエーションのお話。*
- *テーブルの構造から、マルチ・アレイ（Multidimensional Array）の説明をします。*

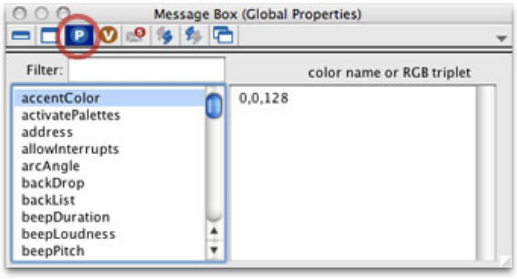
この章で使われる英単語

- template [名詞：型, テンプレイト]
- array [名詞, 動詞：配列（する）]
- element [名詞：要素, 成分]
- combine [動詞：結合する, 組み合わせる]
- split [動詞：分割する]
- cell [名詞：小部屋, 細胞, （表で使う）セル]

オブジェクトのテンプレイト「Template」

LiveCodeは統合開発環境なので、メニュー・ビルダーのように付属のユーティリティの使い方も同時に説明してはいますが、基本的にはプログラミング言語です。もう少しプロパティと、プロパティが構築されている方法について詳しく話しましょう。今までにもプロパティと言う言葉を随所で使ってきました。property（プロパティ）は、もともと財産とか、所有物とかの意味です。日本語のコンピュータの言葉としては、特性とか属性とも言われるようです。LiveCodeで言う「property　プロパティ」は、オブジェクトの見た目の違いや、振る舞いを特徴づけているキャラクター（性質）と、私は捉えています。

メッセージ・ボックスで「the propertyNames」と打ってみると、709行のプロパティが返されます。これはLiveCodeにビルトインされているグローバル・プロパティと言って、アプリケーション全体の見た目や振る舞いに影響をあたえるプロパティです。グローバル・プロパティはメッセージ・ボックスの「Global Properties」で見る事ができます（下図赤丸）。図の例ではアクセント・カラー（accentColor　メニューがアクティブになった時のハイライト・カラー）は、RGB値で「0,0,128」が設定されています。グローバル・プロパティもオブジェクトのプロパティと同じように、呼び出したり、書き込んだりできます。ほとんどのグローバル・プロパティは、オブジェクトのプロパティとは異なるものですが、イメージ・オブジェクトの「paintCompression」や、プレイヤー・オブジェクトの「playLoudness」など、幾つか共通する言葉のプロパティもあります。



オブジェクト個々を特徴づけるほとんどのプロパティは、グローバル・プロパティとは別なプロパティが設定されていて、LiveCodeではオブジェクトの基本となるプロパティを設定している型を、「template　テンプレイト」と言っています。例えば「templateStack　テンプレイト・スタック」とか、「templateGraphic　テンプレイト・グラフィック」とか「templateField　テンプレイト・フィールド」とかがあります。これでオブジェクトの最も基本のプロパティ（properties）を設定していて、コマンドの「Create」によって作られる、オブジェクトの型を決定しているキーワード（keyword）です。

フィールド（field）の基本（テンプレイト）のプロパティ（properties）を調べてみます。
ニュー・メインスタックを「Fileメニュー」から作ってください。今はデフォルトのサイズで良いです。特に名前をつける必要はありません。メッセージ・ボックスからテンプレイト（template）のフィールド（field）を作ります。

```
reset the templateField
create fld 1
```

始めに「reset the templateField」としたのは、何らかの事情でテンプレイト（template）が変わっているとも限らないので、もっとも基本のデフォルトの「templateField」にリセットしました。オブジェクトのテンプレイトは、アプリケーションの開発者が変更を加える事ができます。次の「create fld 1」で、フィールド1（field 1）がカードの中央に作られます。このテンプレイトのテキスト・フィールドのプロパティを聞いてみます。フィールドのプロパティのリストを得るのは

```
put the properties of fld 1
```

これではメッセージ・ボックスに何も返されません。

ここで返されるフィールド１のプロパティーズ（properties）は、アレイ（array　配列）と言われる情報が並んでいるリストなので、一旦アレイ（array）として受け取って、それを解きほぐす「the keys of」と言う言葉と組み合わせなくては、読み取る事ができません。この場合「keys」 はアレイ（array）のリストを構成しているキー（key）の名前を、リターン（return = cr）をデリミッターにして返します。別な言葉で言えば、キーの名前を行毎にして返します。スクリプトで書いてみます。このフィールド（fld 1）のプロパティのリストを受け取るアレイ（array　配列）は、「propArray」と言う名前にしました。

```
put the properties of fld 1 into propArray
put the keys of propArray
```

１行で
put the keys of (the properties of fld 1)
としても同じ結果が得られます。

これで55行のキー（keys）が返されます。この１つ１つのキー（key）に書き込まれている要素（element　エレメント）を取り出してみます。例えばフィールドに縦のスクロールバーがあるかないかのキー「vScrollbar」を例で、エレメント（element）を取り出してみると

```
put the properties of fld 1 into propArray
put propArray["vScrollbar"]   -- アレイの名前の後にブラケット [ ] でキーを括る
```

「false」が返されました。もちろん視覚的にも、縦のスクロールバーはないので、フィールド１（field 1）の「vScrollbar」は「false」である事が確認できます。

この要領で「アレイ名[キー]」として、リピートでひとつづつキーをブラケットに入れて書き出せば、フィールド１に設定されたすべてのプロパティのエレメント（element）がわかるのですが、LiveCodeではこんな場合、コンバイン（combine）を使ってキー（key）と、書き込まれているエレメント（element）を知る方法があります。

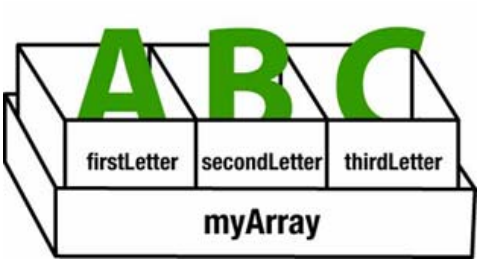
アレイ「array」の基礎

combine（コンバイン）も、key（キー）も、 element（エレメント）も、アレイ（array　配列）で使われるる言葉ですから、簡単なアレイ（array　配列）の例を作って説明しましょう。

バリアブル（variable）は、１つの名前のコンテナーに１つの値が入ります。アレイ（array）は１つの名前のアレイ（array）に複数のエレメント（element　要素）を入れる事ができます。大きな箱のアレイに、キー（key）と言う名前を付けた小さな箱に入れておいて、後からそのキー（key）を目印に引き出す、と考えるのが良いかもしれません。

これからやろうとしているのは、下の図のようにイメージできます。「myArray」と言う名前の大きな箱に、「firstLetter」「secondLetter」「thirdLetter」と言う３つの小さな箱が入っていて、それぞれに「A」「 B」「C」と言う文字を入れます。アレイ（array）の名前は「myArray」。キー（keys）は「firstLetter」「secondLetter」「thirdLetter」。エレメント

（element）は「A」「 B」「C」です。




```
put "A" into myArray["firstLetter"]
put "B" into myArray["secondLetter"]
put "C" into myArray["thirdLetter"]

put the keys of myArray
```

「myArray」と言うアレイの、「firstLetter」と言うキーに、「A」を入れます。
「myArray」と言うアレイの、「secondLetter」と言うキーに、「B」を入れます。
「myArray」と言うアレイの、「thirdLetter」と言うキーに、「C」を入れます。

「myArray」のキー（のリスト）を返しなさい。

上の３行のスクリプトで、「myArray["firstLetter"] 」には「A」が記録されます。同じように「myArray["secondLetter"]」 には「B」が、「myArray["thirdLetter"] 」には「C」が記録されます。これで「myArray」という名前のアレイ（array）が作られます。最後の行、アレイ「myArray」のキー（keys）は「the keys of myArray」で

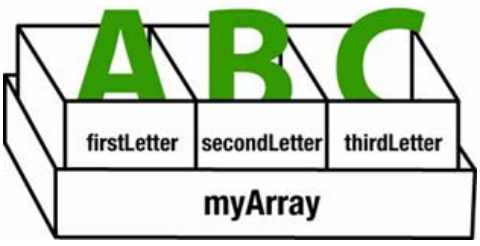
```
firstLetter
thirdLetter
secondLetter
```

が返されます。この３行が「myArray」の「the keys（キー）」です。「the keys」を求めた場合、必ずしもアルファベット順の表記はされません。アレイ「myArray」のキー「firstLetter」のエレメント（element）は、

```
put myArray["firstLetter"]
```

アレイ「myArray」のキー「firstLetter」（と名札のあるエレメント）を返しなさい。

とすれば「myArray["firstLetter"]」のエレメント（element）の「A」が返されます。



もう一度図を見れば、スクリプト「myArray["firstLetter"]」の意味が分かるでしょう。大きな「myArray」と言うアレイの中の、ブラケットで囲われた「["firstLetter"]」と言うキーの中にある文字の事です。

この一連のスクリプトは、メッセージ・ハンドラーを抜けるとアレイ（array）はアレイではなくなります。もしこのアレイ（array）を、他のハンドラーやカスタム・コマンド、カスタム・ファンクション内でも使うようでしたら、アレイ（array）の名前を、同じオブジェクト内ならローカル（local）の宣言、オブジェクト間をまたいで使う場合は、グローバル（global）の宣言をしなくてはいけません。この章の最後に具体例があります。今は説明だけです。

```
global gMyArray, gMyGlobal  -- ２つのグローバルを宣言
```

グローバル（global）の宣言はこのように１行でわかりやすいように、グローバルの言葉を使うオブジェクトのスクリプト・エディターのトップに書く事が多いです。複数のグローバル（global）を宣言するのは、カンマで繋ぎます。LiveCodeディベロッパーの間では、グローバルと判別しやすいように、始めのキャラクターを「g」にする人が多いです。これはプログラミング上のルールではありません。

アレイ（array）からエレメント（element）を取り出すのは、アレイ（array）の名前とキー（key）の組み合わせを１つ１つ行う以外に、コマンドの「combine　コンバイン」を使う方法があります。英語の「combine」は、組み合わせて一体にする意味です。LiveCodeのコマンド「combine」は

```
combine アレイの名前 using デリミッター１ and デリミッター２
```

のように使って、キー（key）とエレメン（element）とをデリミッター（deleimiter　区切り記号）で繋いで、結合させます。具体的な例を見た方がわかりやすいでしょう。デリミッター１が「return」で、デリミッター２は「=」です。下の赤文字を読んでください。

```
put "A" into myArray["firstLetter"]
put "B" into myArray["secondLetter"]
put "C" into myArray["thirdLetter"]   -- ３つのキーとエレメントのアレイ
combine myArray using return and "="
put myArray
```

（訳は４行目だけです）
リターン（return）とイコール（=）を使って、
アレイ「myArray」のキーとエレメントを連結させなさい。

コンバイン（combine）された結果は以下のように

```
firstLetter=A
secondLetter=B
thirdLetter=C
```

「キー（key）　＝ エレメント（element）」が、キーの数だけリターンされて返されます。
combie（コンバイン）で、デリミッターをカンマ（comma）１つだけを使うと

```
combine myArray using comma
```

結果は以下のように、エレメントだけがコンバインされて（繋げられて）返されます。

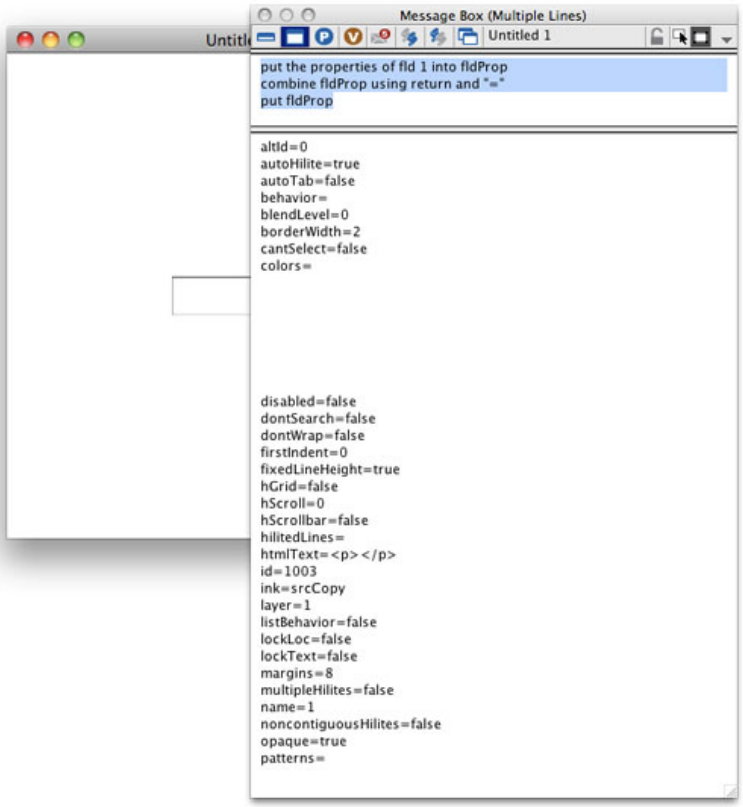
```
A,B,C
```

コンバイン「combine」でフィールドのプロパティ

それでは始めに作ったフィールドのテンプレイトに戻って、フィールドのプロパティ（properties）をキー（keys）にしてそれぞれのエレメント（element）を、コンバイン（combine）を使って書き出してみます。アレイ（array）の名前は「fldProp」にしました。

```
put the properties of fld 1 into fldProp  --「fldProp」でアレイを受け取る
combine fldProp using return and "="
put fldProp        --  「fldProp」はもうアレイ（array）ではありません
```

これでテンプレイト・フィールド（templateField）に設定されているプロパティ（properties）が、イコールとリターンで連結されて書き出されます（下図）。ただし「colors　カラーズ」と「patterns　パターンズ」は、空白の行になっています。ここには「colors」の中にあるフォーグラウンド・カラー（foregroundColor）やバックグラウンド・カラー（backgroundColor）等をキー（key）とした、インスペクターの「Colors & Patterns」にあるプロパティが来るのですが、それにはもう一段踏み込むスクリプトが必要です。しかしどちらにしても現在フィールド 1（field 1）は、テンプレイト・フィールド（templateField）なので、カラー設定は何もされていないので空白になります。



それともうひとつ言っておかなくてはいけないのは、いったんコンバイン（combine）されたアレイ（array）は、もうアレイの形を留めていない普通のテキストになりますから、「the kays」を求めてもエンプティ（empty）が返されます。

```
put the properties of fld 1 into fldProp
combine fldProp using return and "="
put the keys of fldProp
```

エンプティ（empty）がメッセージ・ボックスに返されます。

スプリット「split」で文字列をアレイにする

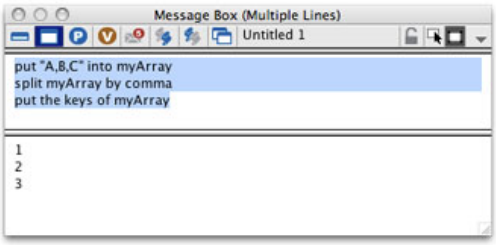
コンバイン（combine）の説明をしたので、スプリット（split）も話しておかなければなりません。英語の「split」は「分離させる」と言う意味で、LiveCodeではコンバイン（combine）とはちょうど反対の役割で、文字列をアレイ（array）に作り変えます。

```
put "A,B,C" into myArray    -- myArrayはまだアレイ（array）ではありません。
split myArray by comma      -- myArrayはアレイ（array）になりました。
```

「A,B,C」を「myArray」に入れなさい。
「myArray」をカンマ（comma）ですスプリット（分離）しなさい。

カンマ（comma）でつないだ文字列をバリアブル「myArray」に入れて、カンマ（comma）をデリミッターとしてスプリット（split　分離）させると、「myArray」と言うアレイ（array）が作られます。この場合「keys」を設定していないので、キー（keys）は数字が順番に割り当てられます。

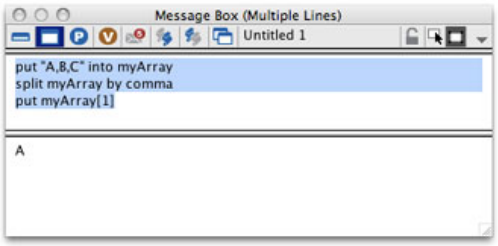
```
put "A,B,C" into myArray
split myArray by comma
put the keys of myArray
```



番号のキー（key）を使って、アレイ（array）からエレメント（element）が取り出せます。

```
put "A,B,C" into myArray
split myArray by comma
put myArray[1]
```

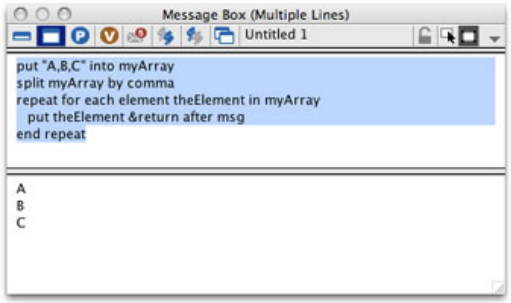
「A」が返されます。



「アレイの基礎」の最後に、リピート（repeat）でエレメント（element）を使ってアレイ（array）のキー（keys）を書き出す方法を書いておきます。

```
put "A,B,C" into myArray
split myArray by comma
repeat for each element theElement in myArray
   put theElement &return after msg
end repeat
```

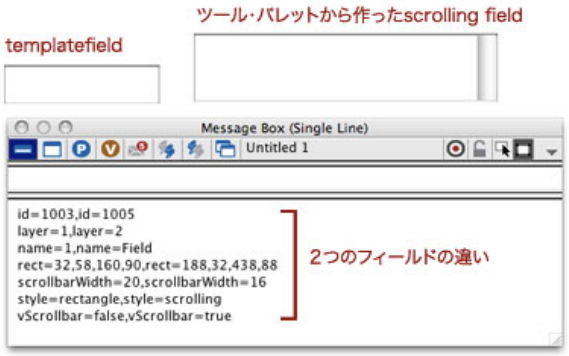
「myArray」のエレメント（element）を「theElemrnt」に入れて、その数だけリピートしなさい
**　　「theElemrnt」とリターンをメッセージ・ボックス（msg）の最後に置きなさい。**
リピート終わり



違うフィールドをテンプレイトで作る

LiveCode初心者では、テンプレイトに手を加えてオブジェクトを作ることはないかもしれません。こういう事もできると言うくらいで読んでください。LiveCodeの学習はガリガリと方法の順序や構文を記憶しないで、こんな事ができると言う、だいたいの索引の様なものを、ゆるく頭に入れておいて、必要な時に後から調べれば良いのです。

これから少し話だけなので、一緒に作ってゆく必要はありません。カードの上にツール・パレットから、スクローリング・フィールド（scrolling field）をドラック＆ドロップして、テンプレイト・フィールド（templatefield）の右に並べて、ふたつのフィールドの「the properties」を呼び出して、プロパティーのエレメント（element）の違いをメッセージ・ボックスに書き出してみました。



カンマで区切られた左がテンプレイト・フィールド（templatefield）で、右がツール・パレットから作られたスクローリング・フィールド（scrolling field）です。「id から rect」まではオブジェクト固有の情報なので、違いがあって当然です。下の３行のエレメント（element）をテンプレイト・フィールド（templatefield）に加えます。「rect」の違いは、「width, height, location」で行います。「name　名前」はコマンドの「create オブジェクト 名前」で付けます。

```
set the scrollbarWidth of the templatefield to 20
set the style of the templatefield to scrolling
set the vscrollbar of the templatefield to true
set the width of the templatefield to 250
set the height of the templatefield to 56
set the loc of the templatefield to the loc of this cd
create field "Field"   -- 名前（name）を「Field」にしました。
```

これでパレットから作るスクローリング・フィールドと同じフィールドが、今開発しているプロジェクトのテンプレイト・フィールドになりました。テンプレイトはリセットされるまでグローバルで内容を保ちますから、もし試していたら必ず「reset the templateField」をしてください。

テーブル・フィールド

ベーシック・テーブル・フィールドはちょっと特殊で、他のオブジェクトでは使われないプロパティの設定方法をとっています。テンプレイトで作ってみます。

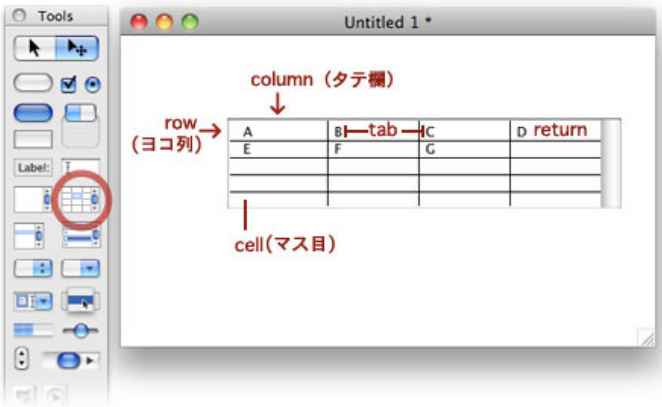
```
set the cRevGeneral["table"] of the templatefield to true
set the cRevTable["celledit"] of the templatefield to true
set the vGrid of the templatefield to true
set the hGrid of the templatefield to true
set the tabStops of the templatefield to 75
set the width of the templatefield to 250
set the height of the templatefield to 60
set the loc of the templateField to the loc of this cd
set the vScrollBar of the templatefield to true
create field "Table Field"
reset the templateField
```

上の２行に「cREV」とあるアレイは、LiveCodeを開発してきたRunRev社がテーブル・フィールドのために、カスタムに作ったアレイのプロパティです。さらにもう一段階複雑にサード・パーティのディベロッパーによって作られたデータ・グリッド（Data Grid）も、ツール・パレットにあります。データ・グリッド（Data Grid）は、ほとんどアプリケーションの使い方の解説になるので、このチュートリアルでは扱いません。

LiveCodeの上級開発者になると、オブジェクトのプロパティをカスタマイズする事で、テンプレイトを発展させて違う形に作り替える事ができます。

マルチ・アレイ「Multidimensional Array」の構造

ベーシック・テーブル・フィールド（Basic Table Field）をカード上に作って、セル（cell）に文字を入れてください。



マウスを使ってセルの１つづつにタイプして行くこともできますが、スクリプトで文字をセットするには、タブ（tab）をコラム（columb　タテ列）の区切りとして、ロウ（row　ヨコ列）の区切りはリターン（return）を使います。

```
put "A" & tab & "B" & tab & "C" & tab & "D" & return & \
       "E" & tab & "F" & tab & "G" into fld "Table Field"
```

テーブルを構成する一組の文字列の構造から、マルチ・アレイ（multidimensional array　多次元配列）を作る事ができます。少し全体的な構成の説明をした後で、シンプルなデータでマルチ・アレイを扱ってみます。

a	b	c
d	e	f

このテーブルをアレイ（array）として書いてみると、タブ（tab）はコラム（columb　タテ列）のデリミッター（columnDelimiter）で、リターン（return）はロウ（row　ヨコ列）のデリミッター（rowDelimiter）になるので、こう書き換えることができます。

```
a <columnDel> b <columnDel> c <rowDel>
d <columnDel> e <columnDel> f
```

このテーブルを「myTable」と言う名前のアレイ（array）として、コラム（column　タテ列）でアレイ（array）をスプリット（split　分離）して、「myTable[1]」を取り出すと、１番目のコラム（タテ列）「a　d」が２行で返されます。

```
put "a" & tab & "b" & tab & "c" & return & \
       "d" & tab & "e" & tab & "f" into myTable
split myTable by column
put myTable[1]
```

タテ１列目のコラム「a　d」が２行で返されます。

```
a
d
```

同じように「myTable[2]」を取り出すと、２番目のコラム（タテ列）「b　e」が２行で返されます。

```
put "a" & tab & "b" & tab & "c" & return & \
       "d" & tab & "e" & tab & "f" into myTable
split myTable by column
put myTable[2]
```

タテ２列目のコラム「b　e」が２行で返されます。

```
b
e
```

同じ「myTable」と言う名前のアレイ（array）を、ロウ（row　ヨコ列）でスプリット（split　分離）して、「myTable[1]」を取り出すと、ヨコ１列目の「a b c」が１行で返されます。

```
put "a" & tab & "b" & tab & "c" & return & \
         "d" & tab & "e" & tab & "f" into myTable
split myTable by row
put myTable[1]
```

ヨコ１列目の「a　b　c」が１行で返されます。キャラクターの間にはタブ（tab）があります。

```
a       b       c
```

```
put myTable[2]    -- 最後の行だけ入れ替えると
```

ヨコ２列目の「d　e　f」が１行で返されます。キャラクターの間にはタブ（tab）があります。

```
d       e       f
```

次に、コラム・デリミッター（columnDelimiter）をタブ（tab）でなく、カンマ（comma）で作ったアレイ（array）から、コラム（column）をひとつを取り出して、普通の３行（3 lines）の文字列に戻します。「cr」は「return」と同じです。

```
-- カンマで区切った文字列を３行「myTable」に入れる
put "a,b,c" &cr& "d,e,f" &cr& "g,h,i" into myTable

-- コラムをデリミッターとしてアレイにする
set the columnDel to comma
split myTable by column
put myTable[2]  -- ２列目のコラムを取り出す
```

以下の３行２列目のコラムが返されます。

```
b
e
h

wait 120  -- ２秒間（120 ticks）待つ

combine myTable using column    -- combine, columnを使って文字列にもどす
put myTable
```

以下の３行が最後に返されます。

```
a,b,c
d,e,f
g,h,i
```

マルチ・アレイ「Multidimensional Array」の使い方

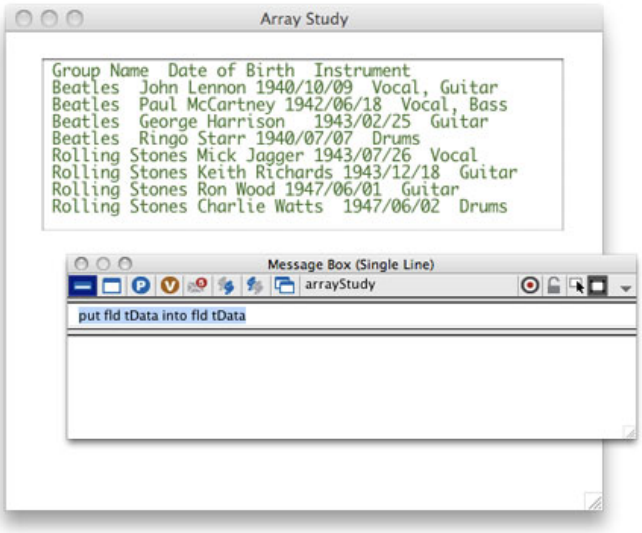
もう少し具体的な例でやってみましょう。BeatlesとRolling Stonesの簡単なテキスト・データがあります。データは、２つのグループ名に大きく分かれて、それぞれのメンバーの名前と、生年月日とメインのインストです。項目は「Group」「Name」「Date of Birth」「Instrument」で、それぞれタブ（tab）で区切られています。つまりタテの項目を分けるコラム・デリミッター（columnDelimiter）はタブ（tab）で、メンバーの横列を分けるロウ・デリミッター（rowDelimiter）は、リターン（return = cr）です。

```
Group   Name    Date of Birth   Instrument
Beatles John Lennon     1940/10/09      Vocal, Guitar
Beatles Paul McCartney  1942/06/18      Vocal, Bass
Beatles George Harrison 1943/02/25      Guitar
Beatles Ringo Starr     1940/07/07      Drums
Rolling Stones  Mick Jagger     1943/07/26      Vocal
Rolling Stones  Keith Richards  1943/12/18      Guitar
Rolling Stones  Ron Wood        1947/06/01      Guitar
```

```
Rolling Stones  Charlie Watts   1947/06/02      Drums
```

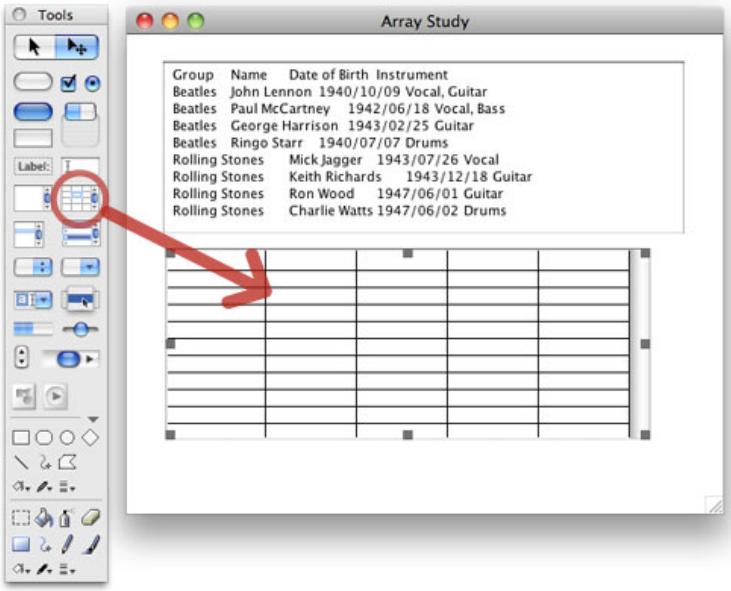
カード上にテキスト・フィールドを作って名前を「tData」とします。フィールド「tData」に、このページから上の緑のテキストをコピペして、メッセージ・ボックスから

```
put fld tData into fld tData
```



と送ると、htmlテキストのタグが除かれて、プレインなテキストがフィールド「tData」に入ります。この作業は必ず必要な作業ではありません。プレインなテキストを確認するためだけのものです。

ツール・パレットからベーシック・テーブル・フィールド（Basic Table Field）をカード上に置いて左右420ピクセルにしてください（下図）。テーブル・フィールドの名前は「tTable」です。

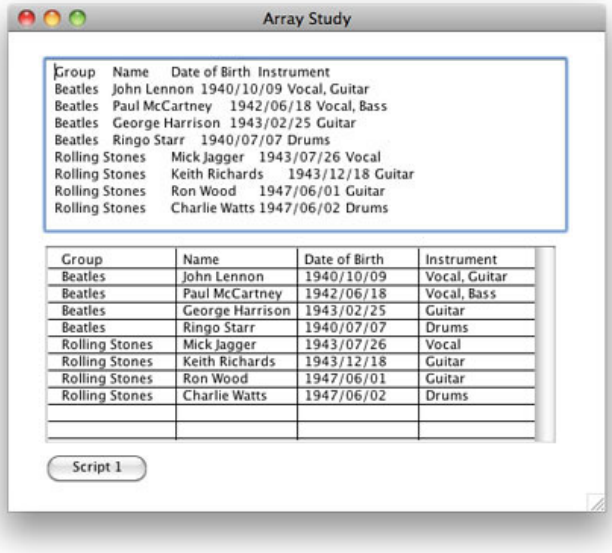


「tData」をテーブルにセットします。ボタン「Script 1」を作って、中のスクリプトは

```
on mouseUp
   set the cRevTable["celledit"] of fld "tTable" to false
   set the tabStops of fld "tTable" to 100
   put fld "tData" into fld "tTable"
end mouseUp
```

オン　マウスアップ
フィールド「tTable」の（編集ができないように）cRevTable["celledit"]をfalseに設定。
フィールド「tTable」のセルの巾を100ピクセルに設定
フィールド「tData」をフィールド「tTable」に入れなさい。
エンド　マウスアップ

デフォルトのベーシック・テーブルは、マウスでカーソルを入れて、セル編集ができるようになっているので、始めの行でできないようにします。「tabStops」はテーブルの１つ１つのセル（cell　マス）の、横巾のサイズです。コラムを区切るタテの線の位置と捉えてもよいです。デフォルトのテーブルのセル巾（tabStops）は75ピクセルです。左右420ピクセルのテーブルの、タブストップ（tabStops）を100にすると、左右４コマで20ピクセルのスクロールバーがあるテーブルになります。そこにフィールド「tData」のテキストを流し込みます。



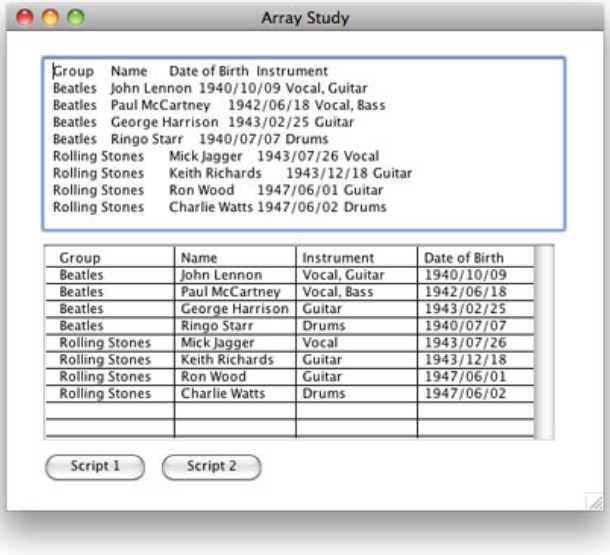
結果は上の図のように、左右４コマのセルのテーブルにコラムの頭揃えでデータが入ります。始めにコラムをそっくり入れ替える事をやってみましょう。３番目のコラム「Date of Birth」と４番目の「Instrument」を入れ替えます。もうひとつ新しいボタン「Script 2」を作ってください。ボタン「Script 2」に入るスクリプトは、

```
global gColumnData, gRowData  -- 他のオブジェクトでも使えるようにグローバル宣言
on mouseUp
   put fld "tTable" into tData  --フィールド「tTable」を「tData」に入れる
   split tData by column  -- 「tData」をスプリットしてコラム毎のアレイにする
   put tData[3] into tTempo  -- コラム[3]（Date of Birth）を「tTempo」に入れる
   put tData[4] into tData[3] -- コラム[4]（instrument）をコラム[3]に入れ替え
   put tTempo into tData[4]  -- 「tTempo」をコラム[4]に入れ替え
   put tData into gColumnData  -- アレイ「tData」を「gColumnDat」に入れる
   combine tData by column   -- アレイ「tData」をテキスト・データに戻す
   put tData into fld "tTable" -- テキスト「tData」をフィールド「tTable」に
```

```
   split tData by row    -- テキスト「tData」をロウ（row）毎のアレイにする
   put tData into gRowData  -- アレイ「tData」をグローバル「gRowData」にする
end mouseUp
```

```
グローバル宣言　gColumnData, gRowData
オン　マウスアップ
    フィールド「tTable」をバリアブル「tData」に入れなさい。
    バリアブル「tData」をコラム（column）でスプリット（split）してアレイに作り替えなさい。
    アレイ「tData」の３列目（tData[3]）をバリアブル「tTempo」に入れなさい。
    アレイ「tData」の４列目（tData[4]）を、\
        アレイ「tData」の３列目（tData[3]）と入れ替えなさい。
    バリアブル「tTempo」をアレイ「tData」の４列目（tData[4]）と入れ替えなさい。
    アレイ「tData」をグローバル「gColumnDat」に入れなさい。
    アレイ「tData」をコラム（column）でコンバイン（combine）してテキストに戻しなさい。
    テキスト「tData」をフィールド「tTable」に入れなさい。
    テキスト「tData」をロウ（row）毎にスプリット（split）してアレイにしなさい。
    アレイ「tData」をグローバル「gRowData」に入れなさい。
エンド　マウスアップ
```

上記スクリプトを「--」の付いている日本語は取り除いて、ボタン「script 2」に入れてボタンのクリックで実行します。下の図のように、コラムの３番目と４番目がそっくり入れ替わります。始めの行で「グローバル宣言　gColumnData, gRowData」をしたのは、これからデータを取り出す作業をするのですが、アレイはハンドラーを抜けるとアレイではなくなるので、次のスクリプトでもアレイ「gColumnData」と、アレイ「gRowData」が使えるように、グローバルにしています。しかしグローバル宣言をしたアレイでも、コンバイン（combine）をすると、アレイではなくなりますがらご注意ください。



それではこのテーブルから、アレイを使ってデータを取り出します。ボタン「Script 3」を作ってください。 以下のスクリプトがボタン「Script 3」に入ります。

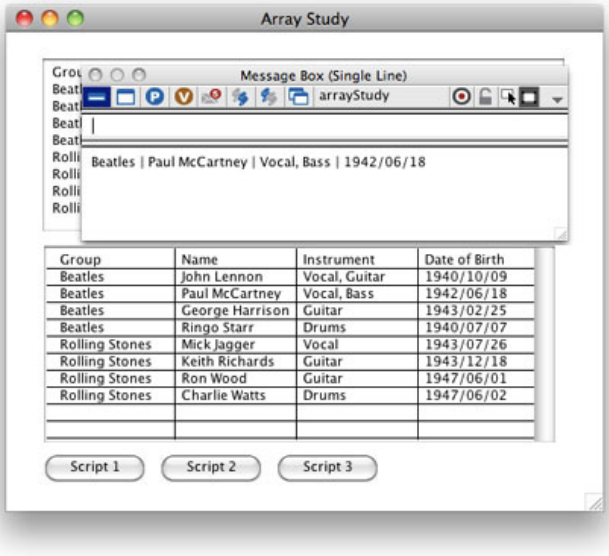
```
global gRowData  -- グローバル宣言 アレイ「gRowData」をこのボタン内で使います。
on mouseUp
   put gRowData[3] into tRow
   replace tab with " | " in tRow
   put tRow
```

```
end mouseUp
```

```
グローバル宣言　gRowData
オン　マウスアップ
    アレイ「gRowData」の３列目（gRowData[3]）をバリアブル「tRow」に入れなさい。
    「tRow」の中のタブ（tab）を「 | 」と入れ替えなさい。
    「tRow」をメッセージ・ボックスに書き出しなさい。
エンド　マウスアップ
```

「gRowData[3]」で取り出して、バリアブル「tRow」のデータはタブ（tab）で区切られていますから、ユーザーに分かりやすい形を取るために「| （タテ棒）」に入れ替えました。普通はコンマ（,）でも良いですが、インストルメントの区切りで使っているので、アイテムを分ける印としてここでは「| （タテ棒）」にしています。書き出した結果は以下です。

```
Beatles | Paul McCartney | Vocal, Bass | 1942/06/18
```



この上では、テキスト・データをスプリット（split）した時に割り当てられた、数字のキー（key）からアレイのデータを引き出しましたが、今度は「rockBand」と言うアレイを作って、グループ名（Group）と、名前（Name）を２つのキーにして、データを引き出します。ボタン「Script 4」を作ってください。

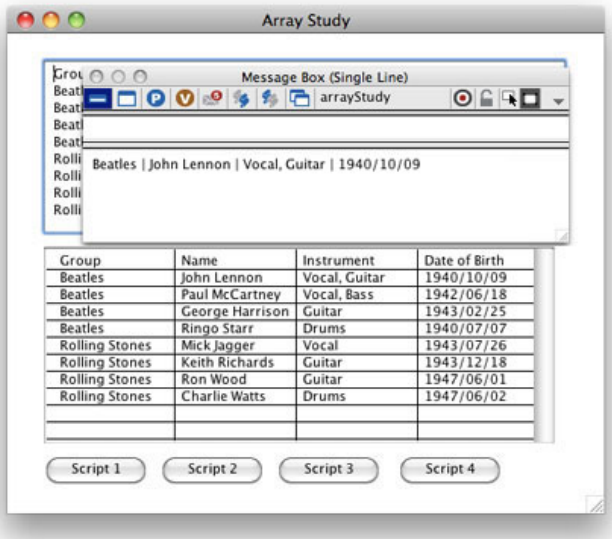
```
global gColumnData  -- グローバル宣言 アレイ「gColumnData」を使います
on mouseUp
   put gColumnData into tData
   combine tData by column   -- アレイ「tData」をコラムでコンバインしてテキストに
   set the itemdel to tab   -- アイテムデリ（区切り記号）をタブ（tab）に
   repeat for each line theLine in tData
      put item 1 to -1 of theLine \
         into rockBand[item 1 of theLine][item 2 of theLine]
   end repeat   -- アレイ「rockBand」を作るすべての作業が終了
   put rockBand["Beatles"]["John Lennon"] into groupAndName
   replace tab with " | " in groupAndName
   put groupAndName
end mouseUp
```

```
グローバル宣言　gColumnData
オン　マウスアップ
    アレイ「gColumnData」を「tData」と言う名前のアレイに移し替える
    アレイ「tData」をコラム（column）でコンバイン（conbine）してテキスト「tData」に変換
    アイテムデリ（区切り記号）をタブ（tab）に設定
    テキスト「tData」の各行をバリアブル「theLine」にして、行数だけリピートしなさい。
        アイテム１から−１までの「theLine」を ¬
            rockBand[アイテム１のtheLine][アイテム２のtheLine]に入れなさい。
    リピート終わり
    rockBand["Beatles"]["John Lennon"] を「groupAndName」に入れなさい。
    「groupAndName」の中のタブ（tab）を「 | 」と入れ替えなさい。
    「groupAndName」をメッセージ・ボックスに書き出しなさい。
エンド　マウスアップ
```

グローバルのアレイ「gColumnData」は、このスクリプトでコンバイン（combine）されると、アレイではなくなってしまうので、グローバルで何度でも使えるようにローカルの「tData」に中身を移します。アレイ「tData」はコンバインされて、タブ（tab）をデリミター（itemDelimiter　区切り記号）とするテキストになります。

テキスト・データ「tData」の行の数だけリピートは繰り返されて、アレイ「rockBand」が作られます。リピートの途中経過をいくつか書きました。
２度目のリピート「rockBand["Beatles"]["John Lennon"] 」には、タブで区切られた
　「Beatles John Lennon Vocal, Guitar 1940/10/09」が入れられます。
３度目のリピート「rockBand["Beatles"]["Paul McCartney"] 」には、タブで区切られた
　「Beatles Paul McCartney Vocal, Bass 1942/06/18」が入れられます。
６度目のリピート「rockBand["Rolling Stones"]["Mick Jagger"] 」には、タブで区切られた
　「Rolling Stones Mick Jagger Vocal 1943/07/26」が入れられます。

もし他のスクリプト中でもアレイ「rockBand」を使う場合は、アレイ「rockBand」をグローバル（global）宣言しておきます。



スクリプトで書き出される結果は、

```
Beatles | John Lennon | Vocal, Guitar | 1940/10/09
```

コラムによって、タブ・ストップの位置を変える

テーブルの最後にタブ・ストップ（tabStops）の細かな設定を書いておきます。上の例ではすべて同じ100ピクセルの巾にしました。しかしそれぞれのコラムによって、もう少し調整したい事も出てきます。タブ・ストップ（tabStops）は、左のマージンからのピクセル数をカンマで区切って設定できます。始めのコラム巾を「100」、次のコラム巾を「120」、次は「90」その次も「90」と設定したいとします。

```
set the tabstops of fld "tTable" to 100, 220, 310
```

これで調整できました。「３カ所しか設定してないじゃないか」と思うでしょう？設定した最後のコラム巾が繰り返しとなるので、同じタブ・ストップ（tabStops）は必要ないのです。タブ・ストップ（tabStops）は複数ですから間違えないようにしてください。

タブ・ストップ（tabStops）と似ている言葉で、タブ・ウイズ（tabWidths）でも、やはりコラム巾の調整ができます。こちらはそれぞれのコラム巾をピクセル数で設定します。やはりこちらもタブ・ストップ（tabStops）と同じに、コラムで区切った最後の値が繰り替えしで、残りの設定されていないコラムの巾になります。

```
set the tabWidths of fld "tTable" to 100, 120, 90
```

このスクリプトでタブ・ストップ（tabStops）で設定したコラム巾と同じになります。タブ・ウイズ（tabWidths）も複数です。どちらでも、書いているコードの流れで使いやすい方を使ってください。

Tips

- アレイ（array　配列）は一連のキー（key）のリストから構成されていて、それぞれの内容を取り出すには、アレイの名前にキーをブラケット[キー]で囲って引き出します。
- 「the keys of アレイの名前」ですべてのキーを取り出しても、アルファベット順の表示になるとは限りません。
- アレイ（array）はハンドラーを抜けると、アレイではなくなるので、他のハンドラーやオブジェクトで同じアレイを使うには、ローカル（local）かグローバル（global）の宣言が必要です。グローバルと分かりやすくするために、始めのキャラクターを「g」にするLiveCode開発者が多いです。
- テーブル・フィールドは、タブ（tab）をコラム（column　タテ列）の区切り、リターン（return）をロウ（row　ヨコ列）の区切りにします。
- アレイ（array）のデリミッターは、タブ（tab）とリターン（return）とは限らず、カンマ（comma）や他のキャラクターも必要に応じて使えます。

この章で新しく出て来た言葉

templateField
キーワード（keyword）新しく作られるフィールドのデフォルトのプロパティ
```
reset the templateField
```

propertyName
LiveCodeにビルトインされているすべてのプロパティのリスト
keys
ファンクション（function）アレイ・バリアブルにあるエレメントの名前のリスト
```
put the keys of propertyNames of field 1
```

element
キーワード（keyword）アレイの中でキーと関連づけて記録されている要素
```
repeat for each element theElement in myArray
```

scrollbarWidth
プロパティ（property）スクローリング・フィールドのスクロールバーの巾
```
set the scrollbarWidth of fld 1 to 20
```

vGrid
プロパティ（property）タブ・ストップで設定されるセルの範囲を決めるタテのグリッド
```
set the vGrid of the fieldTemplate to true
```

hGrid
プロパティ（property）フィールドのテキストの下に表示するライン
```
set the hGrid of the templatefield to true
```

tabStops
プロパティ（property）フィールドのタブをストップさせる位置
```
set the tabStops of the templatefield to 75
set the tabstops of fld "tTable" to 100, 220, 310
```

tabWidths
プロパティ（property）フィールドのコラム巾の位置
```
set the tabWidths of fld "tTable" to 100, 120, 90
```

reset
コマンド（command）オブジェクトのテンプレイトをデフォルトに戻す
```
reset the templateField
```

combine
コマンド（command）アレイをリストに戻す
```
combine fldProp using return and "=" combine myArray using comma
```

split
コマンド（command）リストをアレイに変換する
```
split myTable by column
```

columnDelimiter
プロパティ（property）リストのタテの列で分割するキャラクターを特定する　短縮形：columnDel
```
set the columnDel to comma
```

rowDelimiter

プロパティ（property）リストのヨコの列で分割するキャラクターを特定する　短縮形：rownDel

```
set the rowDelimiter to comma
```

17: ウインドウ・スタックス、スタンドアロン

この章で使われる英単語

- Shape [名詞：形, 輪郭]

この章の概略

- *メインスタックとサブスタックの関係から出て来る問題点と解決法について。*
- *カスタム・プロパティの作り方と引き出し方。*
- *開発プロジェクトの書類をスタンドアロンにする。*
- *もう一度オブジェクト・ハイアラキーについて具体例。*
- *スタック・ウインドウのタイトルバーのデコレーション。*
- *切り抜きウインドウを作ってみる。*

メインスタックは書き換えができません

今までこのチュートリアルは、ほとんどメインスタックばかり使ってやってきました。しかしメインスタックとサブスタックは、幾つかの重要な違いがあります。

第一には、開発しているプロジェクトをスタンドアロンのアプリケーションにした後、メインスタックを書き換えて保存する事ができません。つまりスタンドアロンにしたメインスタックは、開発の最後に作った事柄から書き換えることができません。念のために言っておくと、スタンドアロンと言うのは、開発しているLiveCodeの書類にLiveCodeのエンジンを加えて、開発で使ってきたMac OSではLiveCode.app（WindowsではLiveCode.exe）を開かずに、単独でプログラムが動かせる状態にすることです。スタンドアロンは、他のプログラムの補助なしでも独自で動く、配布可能なプログラムと言う意味です。

ここまで聞くと、大変な事のように感じるかもしれません。しかし短所はいつも長所に切り替わるように、メインスタックを変更して保存はできませんが、サブスタックは変更の保存が可能ですから、それを有利に活用できるプロジェクトの作りを、始めから考えると言うのが、私からのアドバイスです。人によっては、メインスタックをスプラッシュ・ウインドウにすると言うやり方を取る人もいます。スプラッシュ・ウインドウというのは、プログラムが環境設定などのインフォメーションを取り込んでいる間見せておくウインドウで、開発用のLiveCodeアプリで言えば、始めに出る緑のウインドウの事です（下図）。

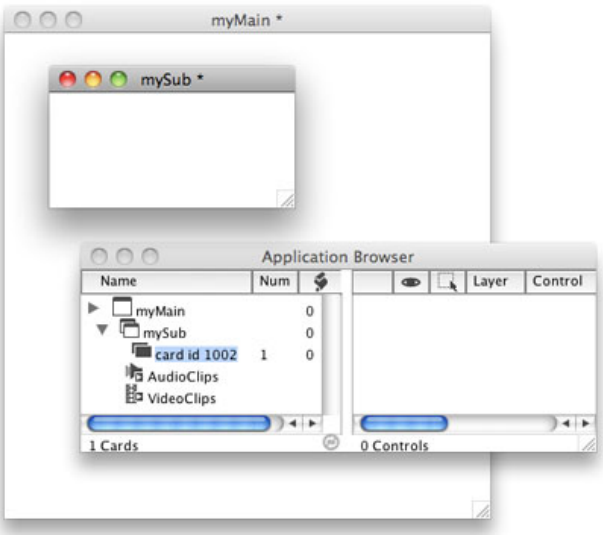
アプリケーションによっては、それも適切かもしれません。しかし必ずしもすべてに適用できる方法でもありません。

他の方法では、メインスタックの変更専用の機能を持たせた、サブスタックを用意すると言うやり方もあります。これはプレファレンセス（preferences　環境設定）のウインドウにして、ユーザーがメニューから引き出させように作ることもできます。

スタンドアロンでは、メインスタックの全ての情報が凍結されるので、メインスタックを開くウインドウの位置もサイズも、開発した時の最後のロケーションに固定されています。これはこれで良い場合もありますが、アプリケーションによっては、前回の使用で最後に閉じた位置やサイズで、メインスタックを開けるのが親切なこともあります。プログラムが閉じられた時の位置とサイズをサブスタックに記録して、その位置にウインドウが開かれるスクリプトを書いてみます。

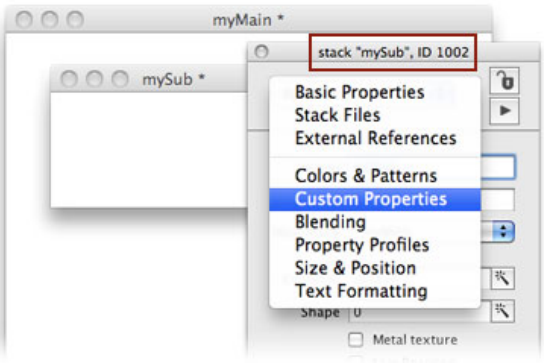
カスタム・プロパティをサブスタックに作る

メインスタックを作って名前（name）は「myMain」、ファイル「myMain.livecode」を保存してください。「Fileメニュー」から「New Substack of myMain」をセレクトしてサブスタック（substack）を作ります。サブスタック（substack）の名前（name）は「mySub」としてください。ちょっとしたテストなので、どちらもスタックのタイトルは付けません。

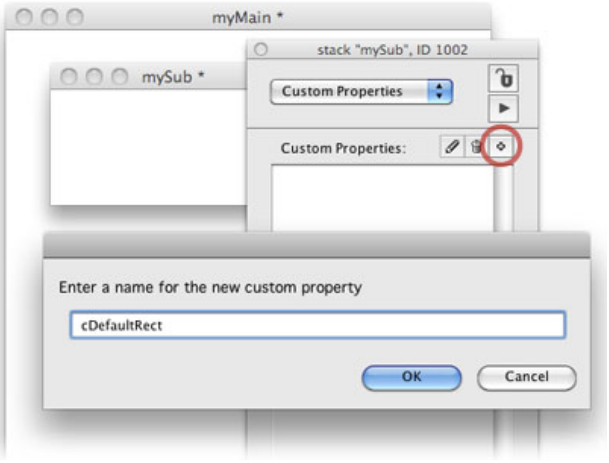


サブスタック「mySub」を環境設定（プレファレンセス　preferences）にも使うのでなければ、ユーザーは開けて見る事はないので、カードには何もコントロールは必要ありません。サブスタックの「mySub」に、書き換え可能なカスタム・プロパティ（custom property）を作って、そこにプログラムが終了した時のメインスタックのロケーション（location）を記録して保存し、次にスタンドアロンが開かれる寸前（preOpenStack）にそれを読み込んで、その位置にメインスタックを開くと言う方法を取ります。

サブスタック「mySub」のカスタム・プロパティ（custom properties）は、インスペクターの「Custom Properties」を開いてください。メインスタックのカスタム・プロパティと間違えないようにしてください。サブスタックでないとスタンドアロンで保存できません。



インスペクターのカスタム・プロパティーズ（Custom Properties）を開いたら、右上の十字印（下図赤丸）をクリックすると、新しく作るカスタム・プロパティの名前をつけるウインドウが現れます。「cDefaultRect」とタイプして「OK」します。カスタム・プロパティの名前はグローバルで「g」を始めに付けたように、始めに「c」を付けるのが、スクリプトを読む時判断しやすいようにと言う、LiveCodeデベロッパーの一般的な命名法です。これはプログラミング上のルールではありません。

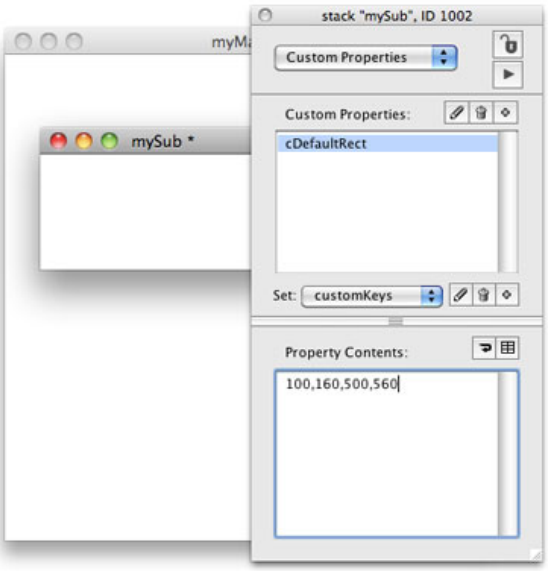


「cDefaultRect」と名前を付けたカスタム・プロパティには、何かのアクシデントに備えて、基本となる現在のスタック「myMain」のレクト（rectangle）を記録しておきます。

```
set the cDefaultRect of stack "mySub" to the rect of stack "myMain"
  スタック「mySub」のカスタム・プロパティ「cDefaultRect」を \
```

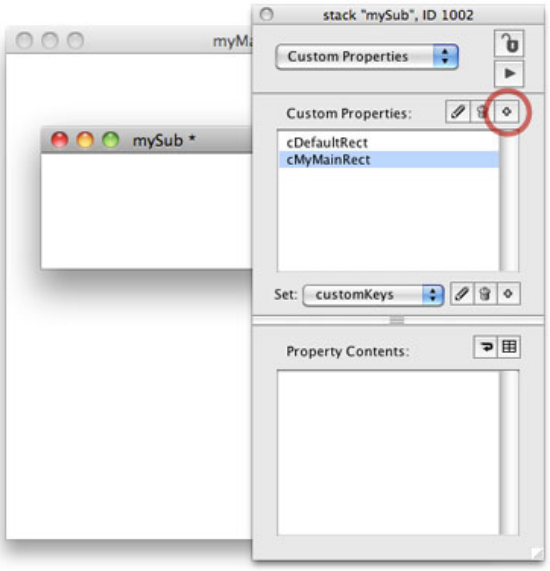
```
スタック「myMain」のレクト（rect）に設定しなさい。
```

とメッセージ・ボックスから送ると、現在のスタック「myMain」のレクト（rect）がカスタム・プロパティにセットされます。一旦インスペクターの「Custom Properties」を他に変えて、もう一度「Custom Properties」に戻らないと、セットした値は見えません。カスタム・プロパティの名前には必ず「the」を付けて、どのオブジェクトに属するか「of オブジェクト名」を付けなければいけません。

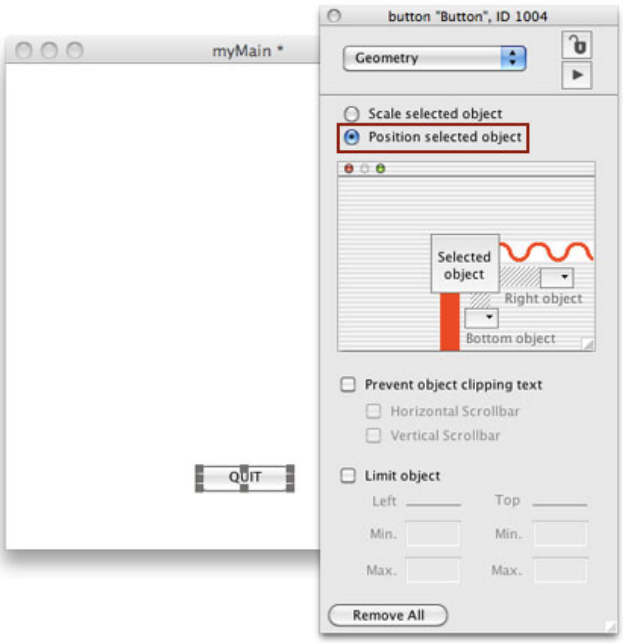


上の図では「100,160,500,560」となっていますが、スタック「myMain」のロケーションによって必ずしもそうなりません。カスタム・プロパティの値は、手動でも修正できます。もし手動で修正したら、左右天地400ピクセルと言う事を確認してください（アイテム３ - アイテム１= 400, アイテム４ - アイテム２= 400）。

もう１つアプリケーションが閉じられる時のレクタングルを記録する、カスタム・プロパティ「cMyMainRect」の名前を作ります。十字印（下図赤丸）をクリックして「cMyMainRect」をタイプして「OK」をしてください。ここにはプロパティ・コンテンツ（Property Contents）を入れる必要はありません。アプリケーションが閉じられる時に、メインスタックのレクタングルを記録させるスクリプトから書き込まれます。



メインスタック「myMain」のサイズが、ユーザーによって変えられると想定して、ボタン「Quit」をメインスタック「myMain」のカードに作ります。ボタンはジオメトリー・マネージャーで下の図のように、ウインドウ・サイズに合わせて移動するようにします。設定し終わったらウインドウ・サイズを変えて試してください。



ボタン「Quit」にプログラムを終了させるスクリプトを書き込みます。

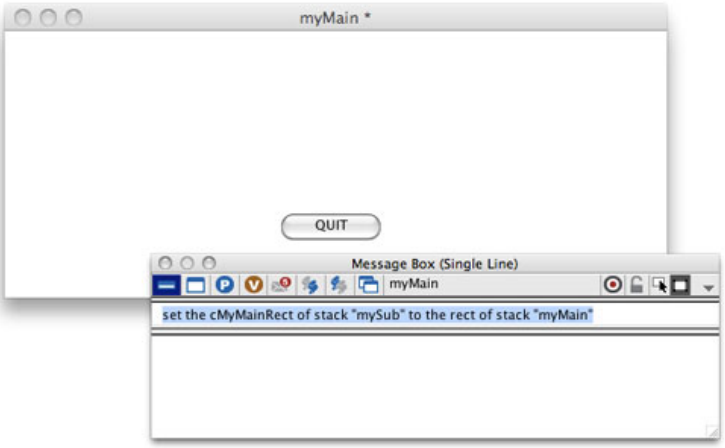
```
on mouseUp
   quit
end mouseUp
```

オン　マウスアップ
**　終了**
エンド　マウスアップ

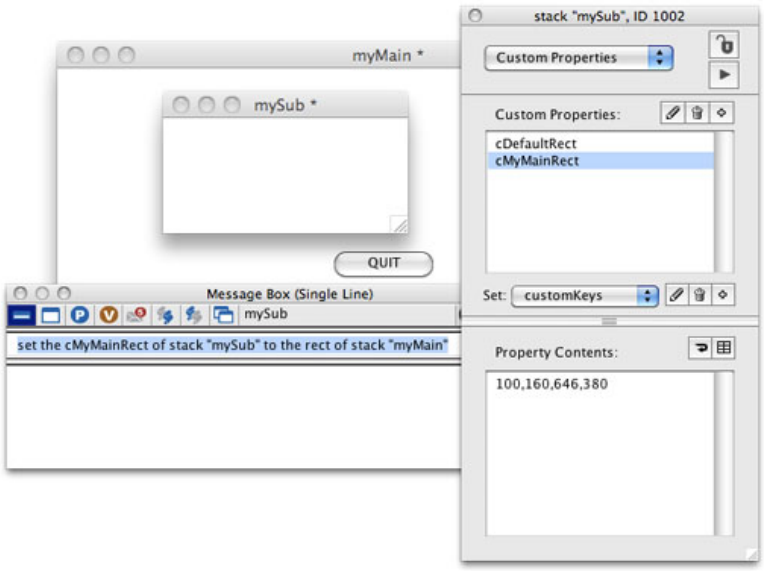
ここで一旦スタックを「Fileメニュー」から「save」してください。

メインスタック「myMain」のサイズ、ロケーションを変えて、サブスタック「mySub」に作ったカスタム・プロパティ「cMyMainRect」に、メッセージ・ボックスから今のレクタングル（rectangle）を書き込んでみます。

```
set the cMyMainRect of stack "mySub" to the rect of stack "myMain"
  サブスタック「mySub」のカスタム・プロパティ「cMyMainRect」を \
      スタック「myMain」のレクト（rect）に設定しなさい。
```

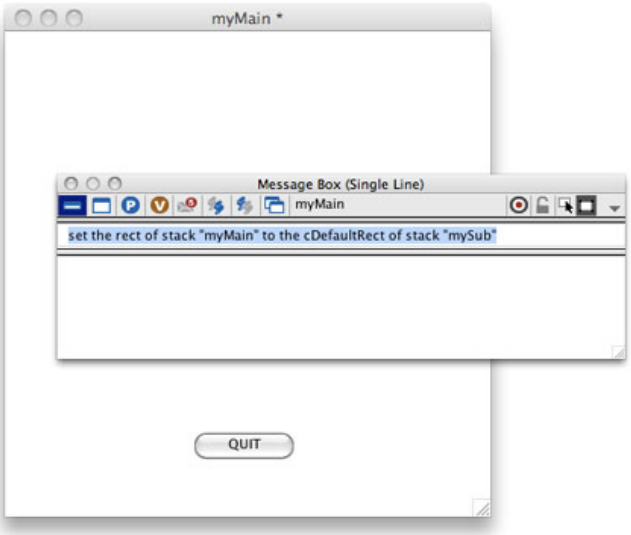


カスタム・プロパティの名前に「the」を付けて送りましたか。サブスタック「mySub」のインスペクターを開いて「cMyMainRect」に書き込まれているか確認してください（下図）。



カスタム・プロパティに書き込んだレクト（rect）から、デフォルトの位置にセットできるか試してみます。

```
set the rect of stack "myMain" to the cDefaultRect of stack "mySub"
  スタック「myMain」のレクト（rect）を \
      サブスタック「mySub」のカスタム・プロパティ「cDefaultRect」に設定しなさい。
```



取り敢えずここまでの内容で、スタック「myMain」のスクリプト・エディターに書き込みます。スクリプトは、プログラムを閉じる時に（closeStack）、メインスタックのウインドウのサイズと位置を、サブスタックのカスタム・プロパティに保存します。次に開く時に（preOpenStack）、そのサイズと位置が、開かれるメインスタックになる内容です。スタンドアロンでそのスクリプトを書いておかない場合は、常に開かれるウインドウの位置とサイズは同じです。

スタック「myMain」のスクリプト・エディターは、「Objectメニュー > Stack Script」。またはMacOSならカードをシフト＋オプション＋コマンド＋右クリック（Windowsはシフト＋コントロール＋オルト＋右クリック）でコンテキスト・メニューを出し、「Edit Stack Script」で引き出せます。

```
-- スタック「myMain」のスクリプト・エディターに書き込む
on preOpenStack    -- スタックが開かれる前に実行する
  set the rect of stack "myMain" to the cMyMainRect of stack "mySub"
end preOpenStack

on closeStack    -- スタックが閉じられる時に実行する
   set the cMyMainRect of stack "mySub" to the rect of stack "myMain"
   save stack "mySub"    -- サブスタック（のカスタム・プロパティ）を保存
end closeStack
```

```
オン　プレオープンスタック
     スタック「myMain」のレクト（rect）を \
        サブスタック「mySub」のカスタム・プロパティ「cMyMainRect」に設定しなさい。
エンド　プレオープンスタック

オン　クローズスタック
     サブスタック「mySub」のカスタム・プロパティ「cMyMainRect」を \
        スタック「myMain」のレクト（rect）に設定しなさい。
     サブスタック「mySub」を保存しなさい。
エンド　クローズスタック
```

日本語部分を除く上の緑のスクリプトを、スタック「myMain」に書き込んでください。

メッセージ「preOpenStack」は、スタック・ウインドウがまだ見えていない時に実行されるスクリプトのハンドラーで、その後メッセージ・ハンドラー「openStack」が実行されます。「openStack」は今は使っていません。

メッセージ「closeStack」は、スタック・ウインドウが閉じられる直前に、必要な整理を実行するスクリプトのハンドラーです。

スクリプトに間違いがないか確認して、スタンドアロンにする前にメッセージ・ボックスから、もう一度メインスタックをデフォルトの設定に戻します。

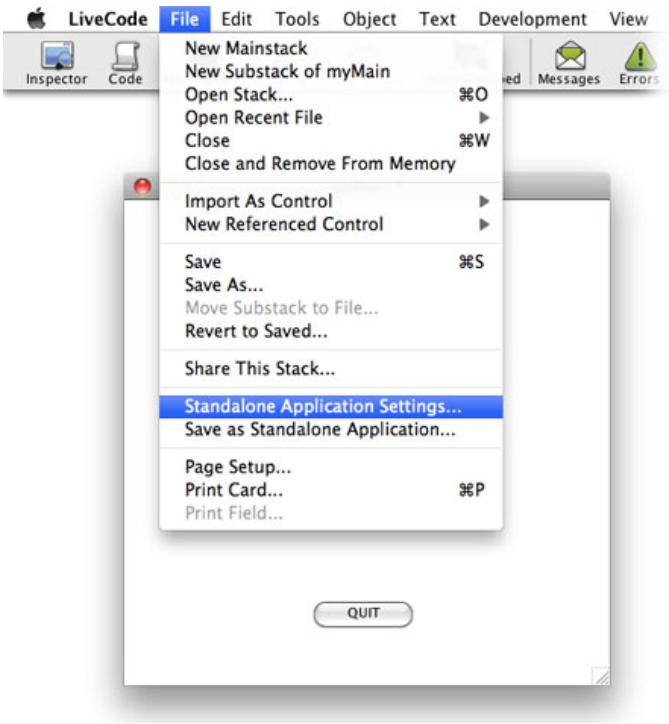
```
set the rect of stack "myMain" to the cDefaultRect of stack "mySub"
set the cMyMainRect of stack "mySub" to the rect of stack "myMain"
save stack "mySub"
```

スタックを「Fileメニュー」から「save」してください。

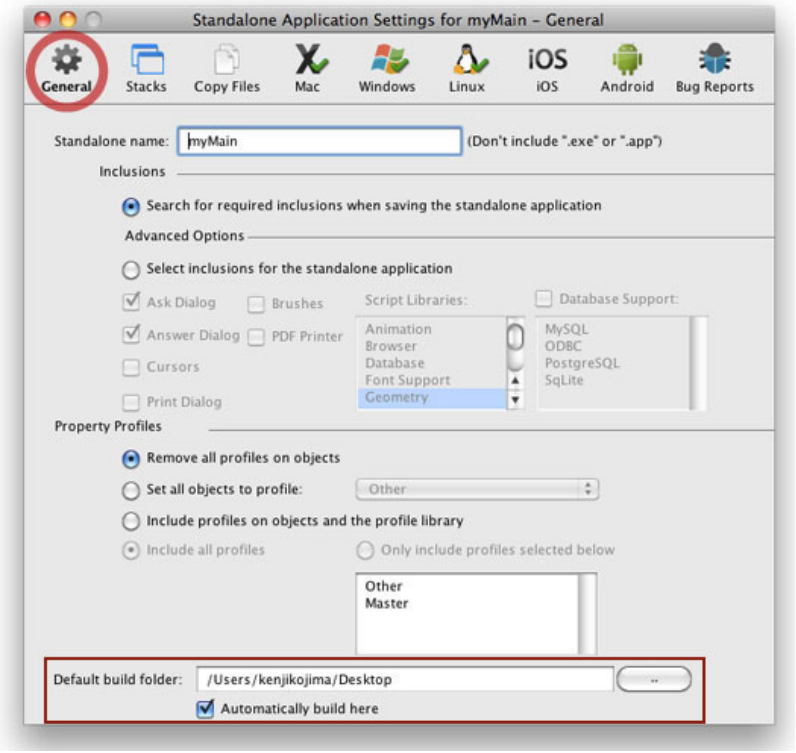
プロジェクトの書類をスタンドアロンにする

ここで試しにスタンドアロンに作ってみます。スタンドアロンは開発の早い段階から、時々作って試しておくというのも、完全に作り上げたスタンドアロンで起きる、思わぬトラブルを避けるLiveCode開発のコツです。

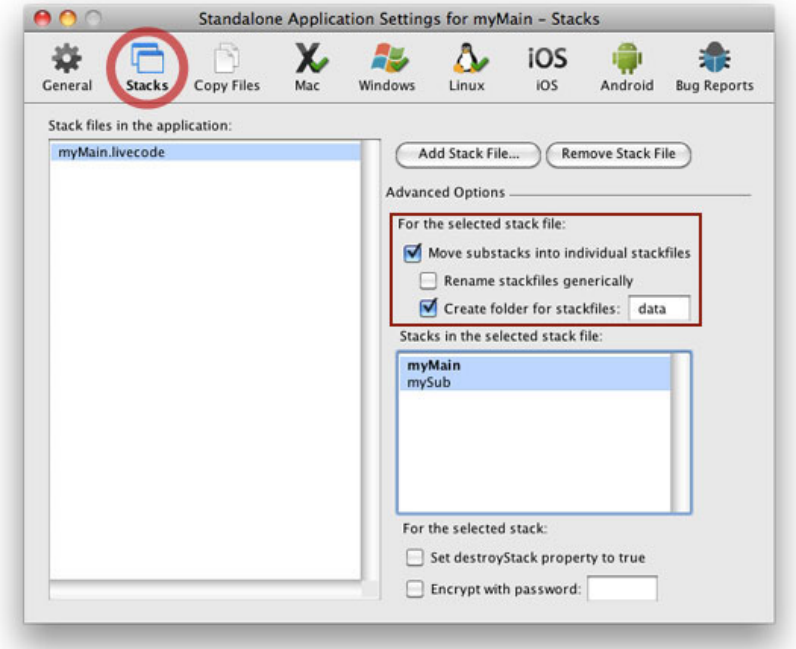
「Fileメニュー」から「Standalone Application Settings...」を選びます。



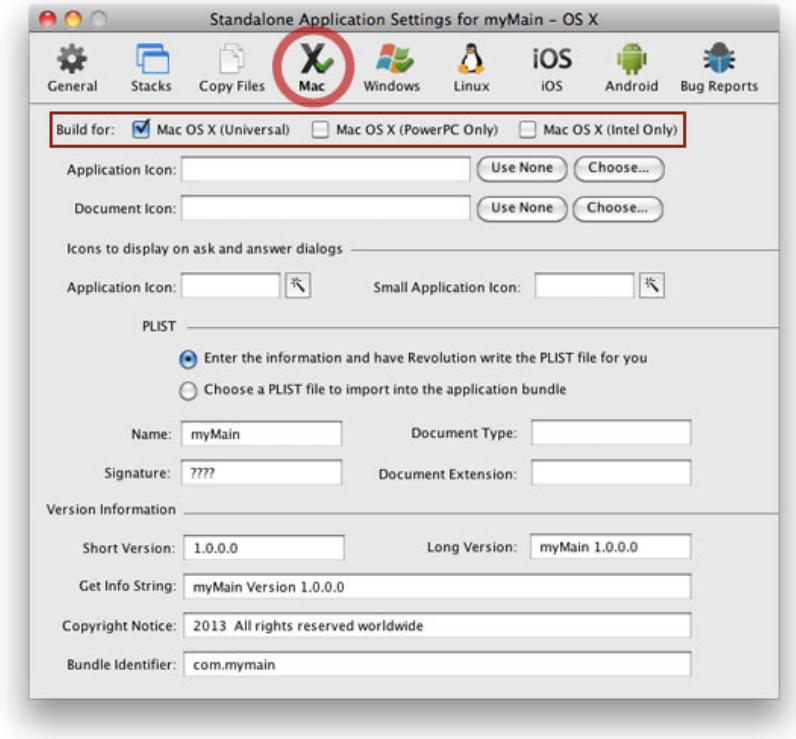
始めに出てくる「General」では、作ったスタンドアロンがデスクトップに現れるように、「Default build folder」をデスクトップにしました（下図赤枠）。



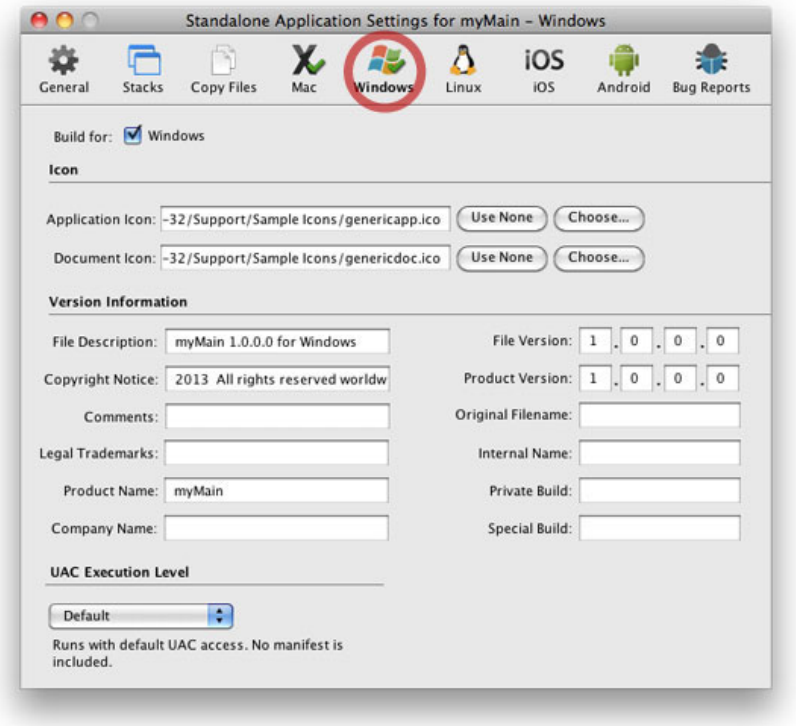
次の「Stacks」は「Move substacks into individual stackfiles」にチェックして、「Create folder stackfiles: data」とすると、サブスタックをまとめて入れるフォルダーが作られます。特にWindowsの場合、サブスタックの書類とアプリケーション・プログラムのアイコンが同じフォルダーに同列に現れるので、複数のサブスタックを「data」と言うフォルダーに１つにまとめます。フォルダー名は「data」でなくても良いです。



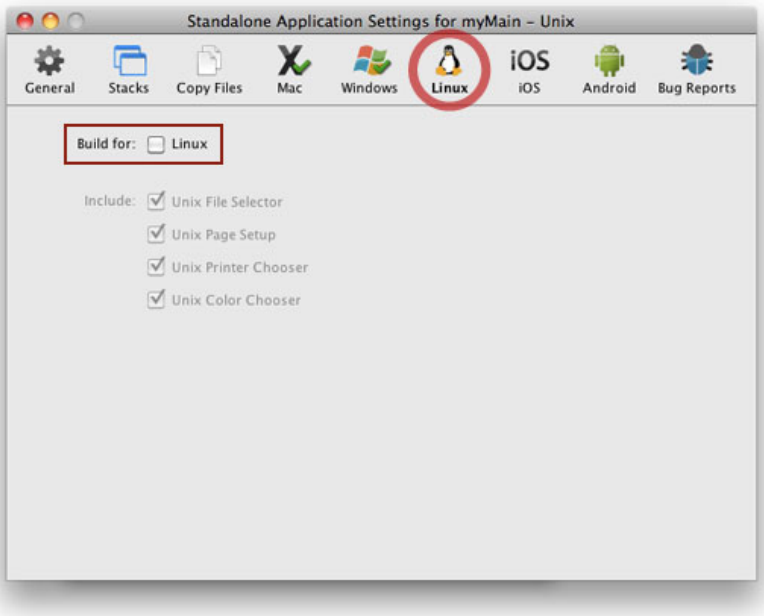
「Mac」は今はテストなので、トップの「Build for:」だけのセッティングをしました。「Mac OS X (Intel Only）」でも良いかと思います。アプリケーション・アイコン（Application Icon）は、空白にしているとデフォルトでLiveCodeのアイコンになります。



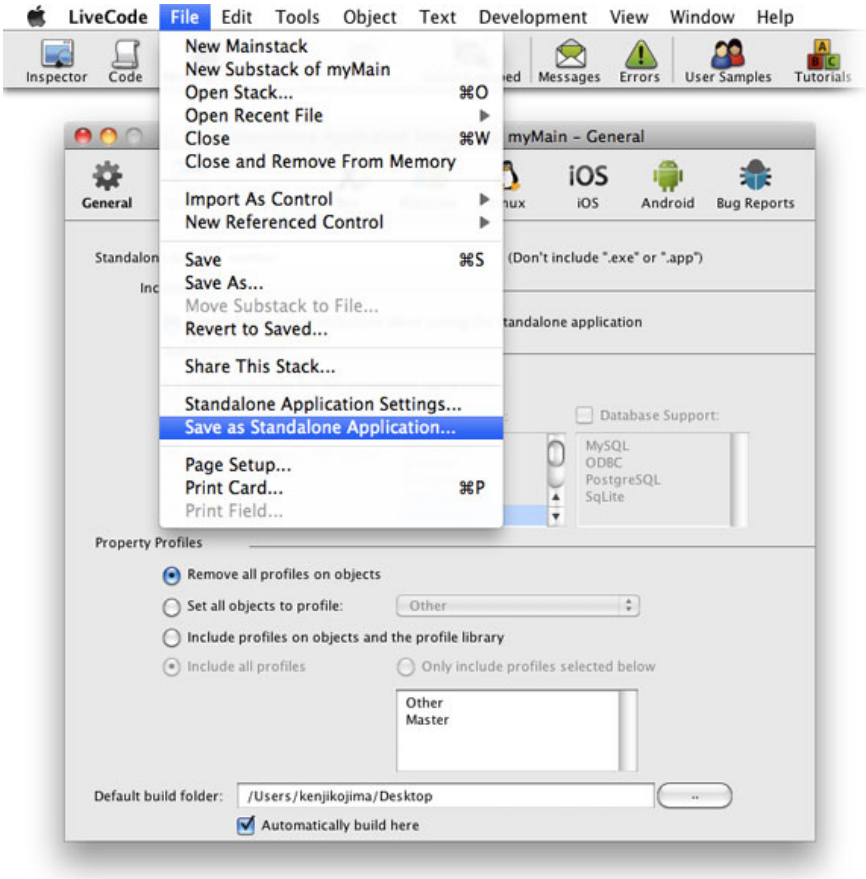
Windowsはデフォルトのままで良いです。



Linuxは作らないので、チェックを外します。



「Fileメニュー」から「Save as Standalone Application...」を選んで、



しばらくすると「Satndalone application saved succesefully.」が出て、デスクトップに「myMain」と言うフォルダーが作られ、中に「MacOSX」と「Windows」のフォルダーがあります。Mac OSは「 myMain.app」のアプリケーション・アイコンが１つだけフォルダーにありますが、Windowsは「myMain.exe」と「data」フォルダーと「Externals」 フォルダーが一緒に入っています。「Externals」は使っていないので、フォルダーは空ですから必要ありません。

「data」フォルダーにはサブスタックの書類「mySub.rev」が入っています。「myMain.exe」と「data」フォルダーは常に一体にして同じフォルダーに入れていないと、アプリケーションは正しく働きません。

Mac OSでは「myMain.app」、Windowsなら「myMain.exe」をダブル・クリックでアプリを開いてスタック・ウインドウのサイズや位置を変えて試してください。正しく働かない場合は、スタック「myMain」に書いたスクリプトに間違いがあるかもしれません。

アクシデントに対処する

それではもう一度、開発環境のLiveCodeから作った書類「myMain.livecode」を開けます。スタンドアロンが正しく働いていたら、次は何かのアクシデントに備えてカスタム・プロパティ「cDefaultRect」を作ったスクリプトを追加します。この場合考えられるアクシデントは、スクリーンの外にスタック・ウインドウが出たまま、戻せないような事が考えられます。人によっては普段ディスプレイを２台つないでいても、場合によって1台と言う事もありますから、それに対処することにしましょう。それともうひとつ単純に、デフォルトに戻すというボタンもカード上に作る事にします。

始めにボタンを作ってメインスタックの左上に置きました。レイベルは日本語で「デフォルトのウインドウに」としました。ついでなのでボタン「Quit」も日本語で「終了」としました。

中に入れるのは、上で書いているスクリプトの１行と同じです。

```
on mouseUp
```

```
   set the rect of stack "myMain" to the cDefaultRect of stack "mySub"
end mouseUp
```

デフォルトのウインドウの位置を修正したかったら、サブスタック『mySub』のインスペクターで直せます。私は「40,110,440,510」にしました。

ディスプレイの外に出た場合のスクリプトは、まずユーザーが使っている現在のスクリーンレクト（screenRects）を調べます。２台のディスプレイにつないでいると、スクリーンレクト（screenRects）は２行のレクタングルを返してきます。「s」を付けて複数形にするのを忘れないでください。「the」を付けろとか、「s」を付けろとか、まあ英語でできているのでこの辺りは厳格です。

```
put the screenRects
```

始めの１行目がメインのディスプレイ、２行目はサブのディスプレイです。もし２行目が空の場合、ディスプレイは１台だけつながっていますから、「the screenRects」の２行目が空白で、しかも「the cMyMainRect of stack "mySub"」のアイテム１が、「the screenRects」の１行目のアイテム３よりも大きければ、明らかにディスプレイの外にスタック・ウインドウがある事になります。スクリプトにする時は、完全にスタック・ウインドウが見えなくなっているよりも、少し余裕をもっていた方が良いので、100ピクセルだけディスプレイサイズからマイナスにしました。

```
-- スタック「myMain」のスクリプト・エディターに書き込む
on preOpenStack
   get the screenRects
   if line 2 of it is empty and (item 1 of the \
        cMyMainRect of stack "mySub" > item 3 of line 1 of it -100 ) then
      set the rect of stack "myMain" to the cDefaultRect of stack "mySub"
   else
      set the rect of stack "myMain" to the cMyMainRect of stack "mySub"
   end if
end preOpenStack
```

```
-- 赤文字だけの訳です。
ディスプレイのサイズをゲットして「it」に入れなさい。
もし「it」の２行目が空で、スタック「mySub」のカスタム・プロパティ「cMyMainRect」の \
        アイテム１の方が、「it」の１行目のアイテム３ - 100 よりも大きかったら
   スタック「myMain」のレクト（rect）を \
      スタック「mySub」のカスタム・プロパティ「cDefaultRect」に設定しなさい。
そうでなかったら
   set the rect of stack "myMain" to the cMyMainRect of stack "mySub"
if文 終わり
```

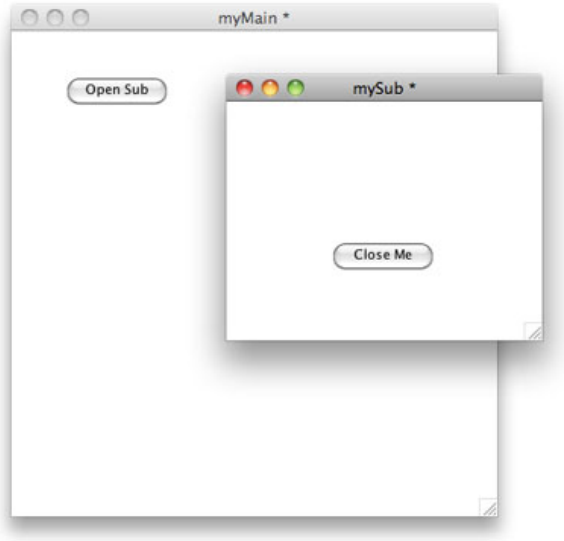
少なくとも私のテストではうまくいきました。

カスタム・プロパティ（custom property）は、サブスタック（substack）だけに作れるとは限りません。すべてのオブジェクトに作って、どこからでも引き出す事ができます。

サブスタックは、メインスタックの支配下にあります

第一に、メインスタックは書き換えを保存できないと書きました。第二に書かなければいけないのは、メインスタックのスクリプト・エディターに書いてある内容は、すべてのサブスタックにも適用されると言う事です。ですからメインスタックのスクリプト・エディターに書き込む内容は、幾つもサブスタックを作った場合、メインスタックと、どのスタックに適用するのか明確に把握している必要があります。特に「preOpenStack」「openStack」に書いたスクリプトをメインスタックだけを意識して書いていると、他のサブスタックを開いた時そのステートメントが実行されて、アレレという事になってしまいます。

また簡単なテストをしてみましょう。メインスタック「myMain」とサブスタック「mySub」を、新しく作ってください。Fileメニューから「Save」して、今度はファイル名を「myMain2.livecode」にします。メインスタック「myMain」はデフォルトのサイズでカード上にボタン「Open Sub」を作ります。サブスタック「mySub」は小さなサイズにしてカード上にボタン「Close Me」を作ります（下図）。




```
-- ボタン「Open Sub」に入れるスクリプト
on mouseUp
   open stack "mySub"
end mouseUp

-- ボタン「Close Me」に入れるスクリプト
on mouseUp
   close stack "mySub"
end mouseUp
```

単純なスタックを開けたり閉じたりのスクリプトです。特に問題なく開けたり閉じたりできると思います。メインスタックのスクリプト・エディターに書き込んだスクリプトが、すべてのサブスタックに影響します。以下をメインスタックのスクリプト・エディターに書き込んでください。

```
-- メインスタックのスクリプト・エディターに入れるスクリプト
on openStack
```

```
   set the width of this stack to 400
   set the height of this stack to 400
end openStack
```

メインスタックにある「openStack」はすべてのサブスタックがオープンされる時にも適用されて、サブスタックが開けられると天地左右は400x400になります。

サブスタックをメインの支配下から逃す

場合によっては、メインスタックとサブスタックの関係は便利に使う事ができますが、時にはちょっと使いにくかったりもします。メインスタックだけの「openStack」にする方法は、メインスタックのカード１に「openStack」のスクリプトを書く事もできます。カードのスクリプト・エディターは「Objectメニュー　＞　Card Script」で開けます。

```
-- メインスタックのスクリプト・エディターのスクリプトはすべて削除して
-- メインスタックのカード１（card 1）のスクリプト・エディターに入れるスクリプト
on openStack
   set the width of this stack to 400
   set the height of this stack to 400
end openStack

-- メインスタックのボタン「OpenSub」に入れるスクリプト
on mouseUp
   open stack "mySub"
   close stack "myMain"
end mouseUp

-- サブスタックのボタン「CLose Me」に入れるスクリプト
on mouseUp
   close stack "mySub"
   open stack "myMain"
end mouseUp
```

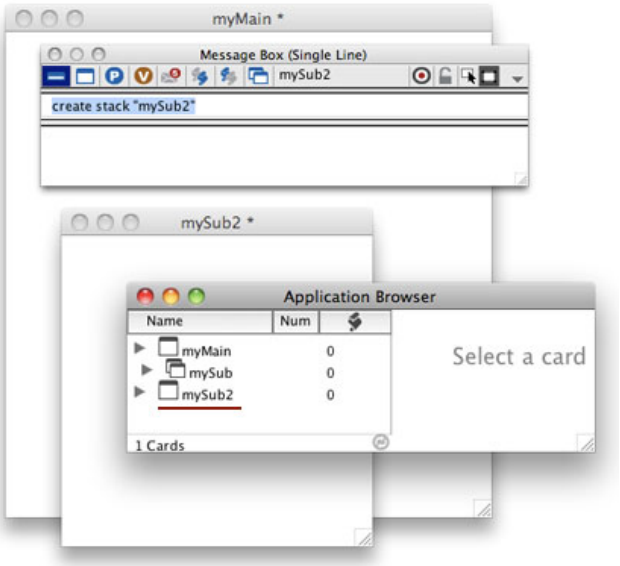
これでメインスタックが開かれる時はいつも「400X400」になります。実際にはメインスタックのロケーション等も考えて、もっと細かな配慮をしたスクリプトが必要です。

サブスタックをクリエイトする

スクリプトでサブスタックを作るには２段階必要です。サブスタック「mySub2」を作ってみます。始めにまずメインスタック「mySub2」を作ります。

```
create stack "mySub2"
```

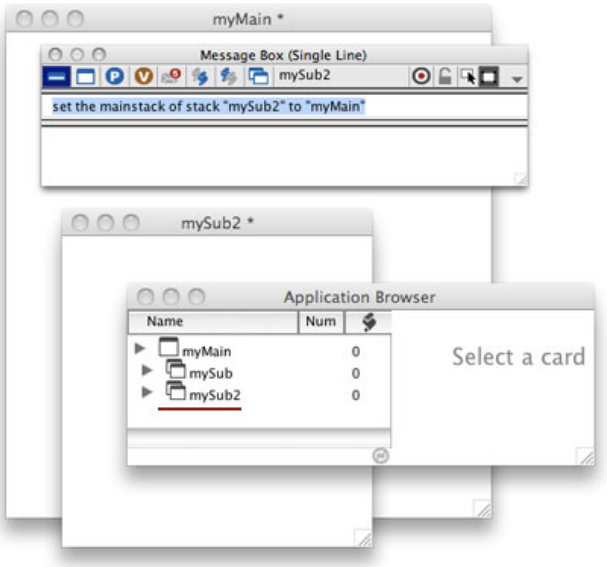
スタック「mySub2」を創りなさい。



上の図アプリケーション・ブラウザー内赤線の「mySub2」を見てください。トップにある「myMain」と同列にあります。これを次のスクリプトで「myMain」のサブスタックにします。と言うより、スタック「mySub2」のメインスタックを「myMain」にと言う表現になるのですが。

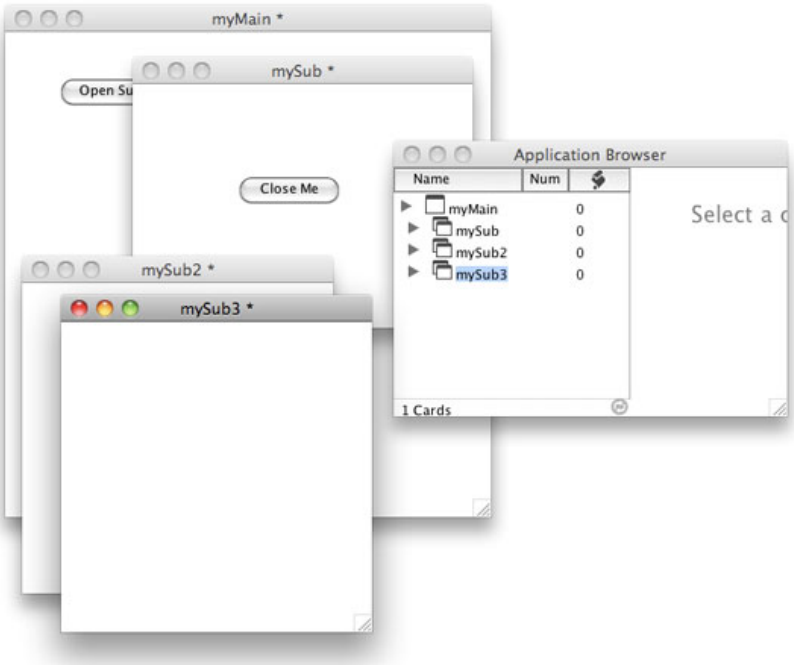
```
set the mainstack of stack "mySub2" to "myMain"
```

スタック「mySub2」のメインスタックを「myMain」に設定しなさい。

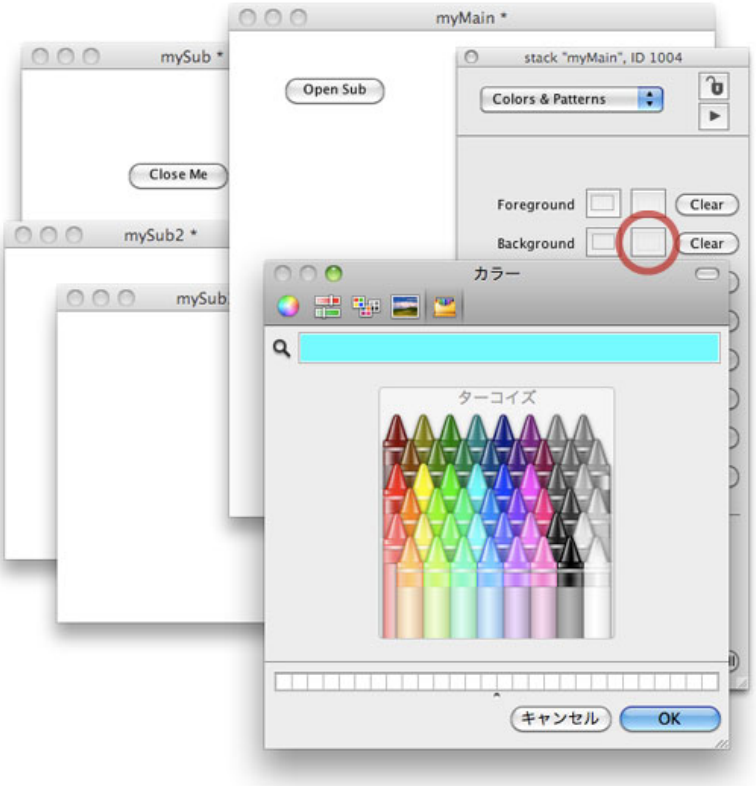


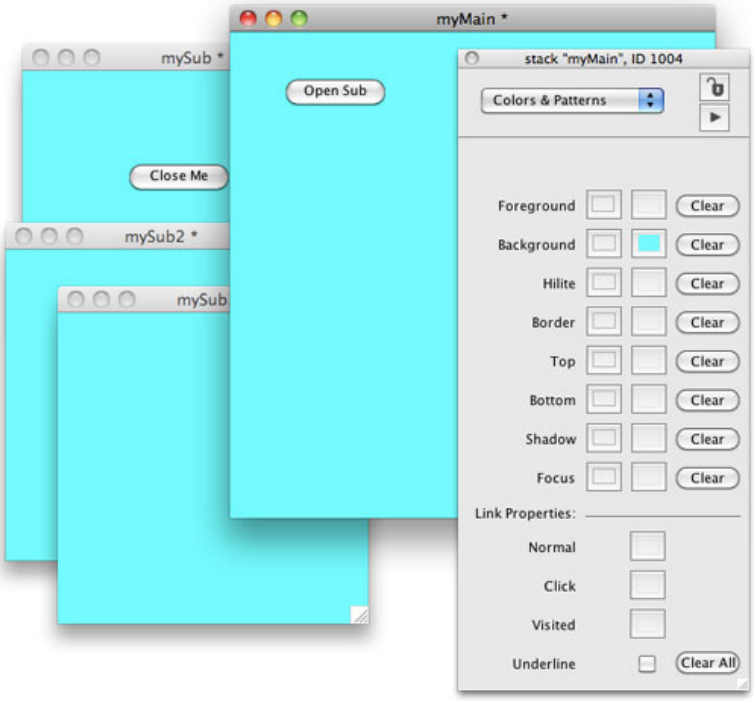
これで、スタック「mySub2」はメインスタック「myMain」のサブスタックになります。もうひとつサブスタックを作ります。

```
create stack "mySub3"
set the mainstack of stack "mySub3" to "myMain"
```



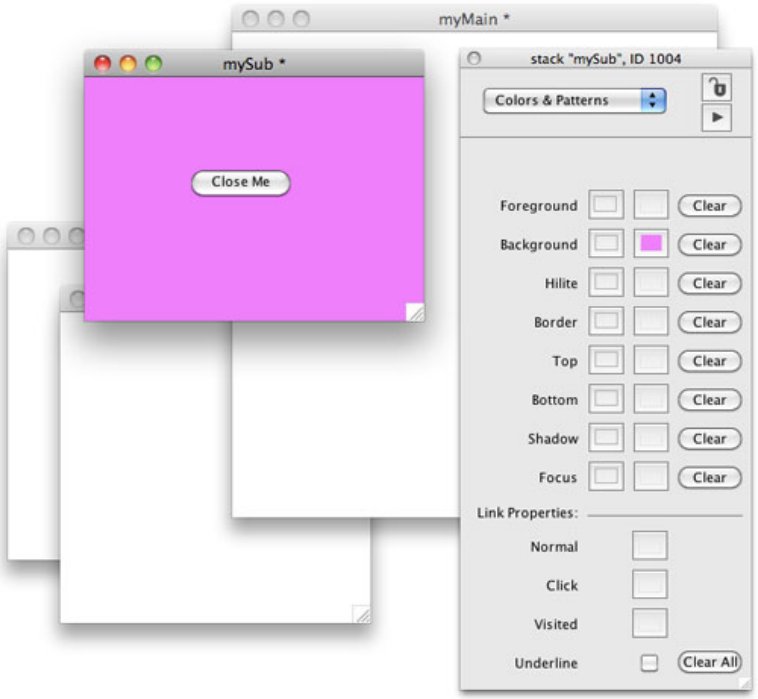
上で「サブスタックは、メインスタックの支配下にあります」と書きました。LiveCodeでは、これを「object hierarchy　オブジェクト・ハイアラキー」と言って、９章でグループ・オーナーの説明をする時にも使いました。階層の上位のオブジェクトのプロパティによって、下位のオブジェクトのカラーやフォントが決められます。ここでメインスタックのバックグラウンド・カラー（backgroundColor）で実験してみましょう。スタック「myMain」をスタックのトップに持ってきて、スタック「myMain」のインスペクターから「Colors & Patterns」でBackground（バックグラウンド）でカラー設定を選んで、何でも好きな色を「OK」してください（下図）。





「OK」をした瞬間にすべてのスタックのバックグラウンド・カラー（backgroundColor）が同じ色に変わります。もしメインスタックのフォーグラウンド・カラー（foregroundColor）の色を設定していたら、すべてのスタックにある文字、テキスト・フィールドもボタンのレイベルも同じ色になります。

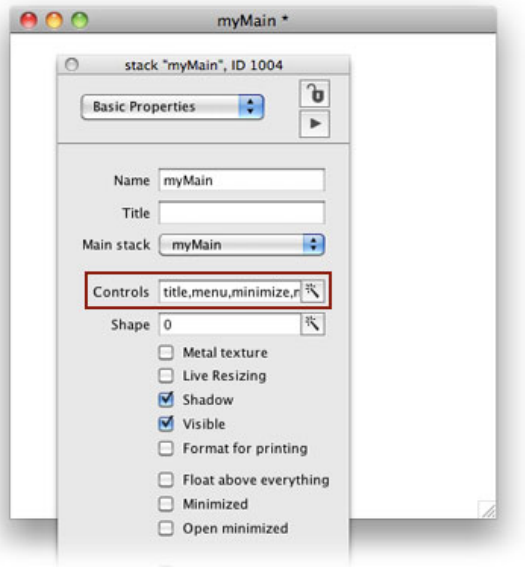
ではサブスタックを選んでバックグラウンド（backgroundColor）を変えてみると、そのサブスタックだけのカラーが変わります。オブジェクト・ハイアラキー（object hierarchy）のオーナー（owner）のプロパティ（property）は、スタック内のカードにも、カード上にあるすべてのコントロールにも及びます。



上にも書いているように、これも使い方しだいで便利にも不便にもなる事で、重要なのは
LiveCodeのこういうクセを予め知っておく事です。

タイトル・バーのデコレーション「Decorations」

必ずしもすべてのプロパティ（properties）にオブジェクト・ハイアラキー（object hierarchy）
が適用される訳ではなく、ウインドウ・タイトル・バーの形式を決めるデコレーション
（decoration）は、スタックそれぞれで決める事ができます。



左の列がMac OSのタイトル・バーのデコレーション、右の列がWindowsのデコレーションです。スタックの必要に応じて変えられます。スクリプトでは下のようになります。クオートの中に入れるコントロールのリストは、図の赤枠を見てください。Mac OSではコントロール・リストの「menu」は、タイトル・バーには見えません。

```
set the decorations of this stack to "title, menu, minimize, close"
```
このスタックのデコレーションを「title, menu, minimize, close」に設定しなさい。

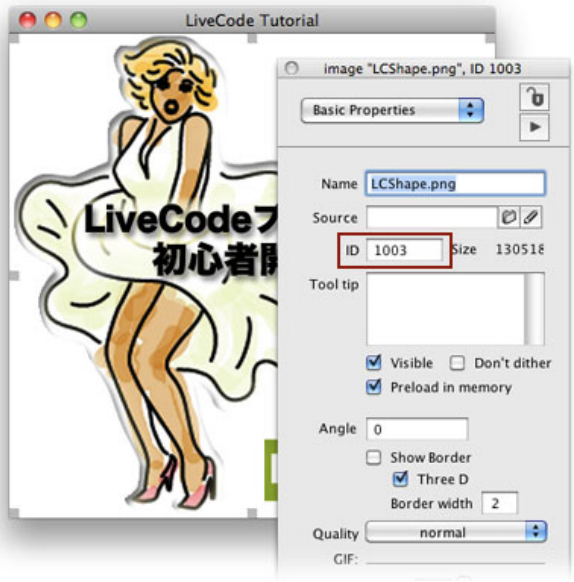
```
set the decorations of this stack to default
```
デフォルトにすると、すべてのコントロール・リストがタイトル・バーに現れます。

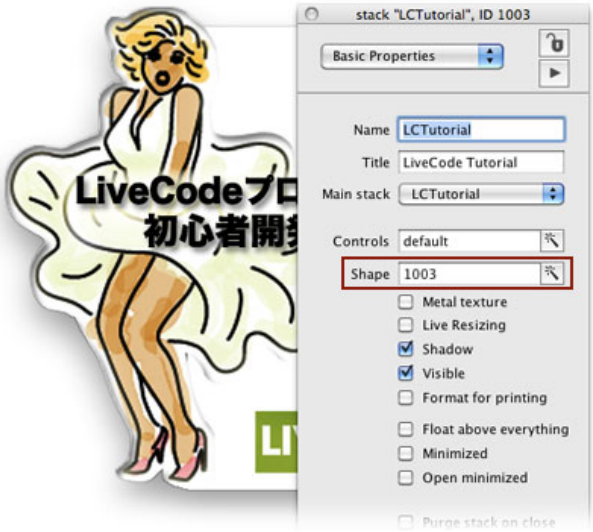
切り抜きウインドウ

アプリケーションを開いた直後に見せる、スプラッシュの切り抜きウインドウのサンプルを作ります。もちろんスプラッシュだけとは限りません。始めにフォトショップなどでPINGファイルを作ってください。下のサンプルPINGファイルは、ダウンロードできます。
http://kenjikojima.com/livecode/download/LCShape.png

これはスクリプトで操作というより、イメージをうまく作れるかなどがより重要になります。形が決まったら、透明 PING が作れるプログラム「Illustrator」や「PhotoShop」などで、PNG-24 透明ファイルを作ります。図のグレーの格子になっている部分は透明です。それ以外の部分がウインドウの形になります。

このPING原稿は400x400で作っていますから、そのサイズのメインスタックを作ります。スタックはリサイズ（resize）できないようにします。メニューの「Fileメニュー　＞　Import As Control　＞　Image File...」で、透明のあるPINGファイルをインポートします。（または透明 GIF でも可）インポートしたら、イメージのプロパティー・インスペクターで「ID」を確認してください。下の図では 1003 がこのイメージのID です。





次に、このスタックのインスペクターのShape（ウインドウシェイプ　windowShape）を、上で確認した「ID」の「1003」にすると、PINGイメージの透明部分が切り抜きとなって、ウインドウのシェイプが現れます。スクリプトでは、こういう表現になります。

```
set the windowShape of this stack to 1003
```

このスタックのウインドウシェイプ（windowShape）を（ID）1003に設定しなさい。

直線で形を作るのはさほど難しくはないのですが、カーブのラインをきれいに切り抜こうとするのは、ちょっとしたコツが必要です。PhotoShop で透明部分をアンチエイリアスで作ったりすると、かなり汚くなります。普通アンチアイリアスの中間トーンは、バックの白地にとけ込んできれいに見えるはずですが、そのあいまいなトーンもすべてウインドウの形になって現れてくるためです。そのあたりの注意をしながら、透明イメージを作ってください。

タイトルバーなしで、動かせるウインドウ

もうひとつ大事なポイントがあります。 切り抜きウインドウを作ると、スプラッシュのように自動的に次のスタック・ウインドウが開かれれば問題ないですが、ウインドウのタイトルバーがないので、どこかにウインドウを動かすハンドルを付けなくてはいけません。ここでは、ウインドウのどこででもマウスで動かせるように、下記のスクリプトを全体を覆っているイメージ内に書き込んでいます。オブジェクト内のすべてのハンドラーで共通に使える ローカル（local）を使っています。

```
-- 透明PINGイメージに入れるスクリプト

local lAllowDrag   -- 「lAllowDrag」をローカル宣言します

on mouseDown
   put the mouseH & "," & the mouseV into lAllowDrag
   -- マウスのXYポジションを「lAllowDrag」に入れます
end mouseDown

on mouseMove x,y
   if lAllowDrag is empty then exit mouseMove
   set topLeft of this stack to \
         globalLoc(x-item 1 of lAllowDrag & "," & y-item 2 of lAllowDrag)
end mouseMove

on mouseUp
   put empty into lAllowDrag
end mouseUp
```

ハンドラー「mouseDown」をした時に、マウスの位置（XポジションとYポジション）をカンマ（, ）でつないで、同じオブジェクトのどのハンドラー内でも使える、ローカルのバリアブル「lAllowDrag」に入れます。「mouseH」はマウスのXポジション、「mouseV」はマウスのYポジションを返すファンクションです。

ハンドラー「mouseMove」では、バリアブル「lAllowDrag」（マウスのXYポジション）がエンプティ（empty）であれば、ハンドラー「mouseMove」から実行が抜けます。それ以外、つまり「lAllowDrag」にXYポジションがあれば、スタックのトップレフト（topLeft）をグローバルロック（globalLoc）に変換した、スタックの「0,0」に移動させます。「lAllowDrag」に入っているXYポジションは、スタックの「0, 0（トップレフト）」から「X,Y」のローカル・ポジションです。「(x-item 1 of lAllowDrag & "," & y-item 2 of lAllowDrag)」は、始めにマウスダン（mouseDown）したポジションからの差を、XYポジションにしています。マウスアップ（mouseUp）されるまで「lAllowDrag」は同じ値です。

```
globalLoc(ローカルのポイント)  -- グローバル・ポイントに変換して返します
```

ハンドラー「mouseUp」が実行されたら、バリアブル「lAllowDrag」をエンプティ（empty）にして、マウスムーブ（mouseMove）から抜けます。

切り抜きウインドウにアプリケーションの機能を作ったら、ウインドウを閉じる設定も必要ですから、このサンプルでは右上にスタックを閉じるボタンを付けました。透明のボタンを切り抜きイメージの中に作った、右上の小さな四角のアイコンの上に乗せて終了ボタンにします。

Tips

- スタンドアロンにすると、メインスタックの変更保存ができないので、それをうまく使えるプロジェクトを始めから考えておきます。
- カスタム・プロパティをサブスタックに保存すると、メインスタックの書き換えも可能になります。
- 開発の初期から時々スタンドアロンを作っておくと、完全に書類が完成してからのトラブルよりも、修正ができてダメージも少なくてすみます。
- スタンドアロンにする時、セッティングの「Stacks」でスタックファイルのフォルダーを作っておけば、複数のサブスタックを１つにまとめられます。
- メインスタックからのオブジェクト・ハイアラキー（object hierarchy）から、サブスタックを逃すひとつの方法は、メインスタックの始めのカードに「preOpenStack」や「openStack」を書く方法があります。
- 切り抜きウインドウの透明イメージをPINGやGIFFで作る時、輪郭をアンチエイリアスにすると、透明部分に影響が出てかえって汚くなる事がありますから、注意してください。

この章で新しく出て来た言葉

closeStack
メッセージ（message）スタックが閉じられる時に現在開いているカードにステートメントを送る

```
on closeStack
  save stack "mySub"
end closeStack
```

mainstack
プロパティ（property）サブスタックが所属するメインスタックを特定する

```
set the mainstack of stack "mySub2" to "myMain"
```

decorations
プロパティ（property）スタック・ウインドウのバーに現れるコントロールを特定する

```
set the decorations of this stack to "title, menu, minimize, close"
```

windowShape
プロパティ（property）ウインドウの形をマスクして変えるイメージを特定する

```
set the windowShape of this stack to 1003
```

mouseH
ファンクション（function）マウス・ポインターの水平点（the Horizontal position）
mouseV

ファンクション（function）マウス・ポインターの垂直点（the Vertical position）

```
put the mouseH & "," & the mouseV into lAllowDrag
```

mouseMove

メッセージ（message）マウスが動かされた時に送られる

```
on mouseMove
  if lAllowDrag is empty then exit mouseMove
  doSomething
end mouseMove
```

globalLoc

ファンクション（function）ローカルの座標をグローバルで返す

```
globalLoc(the mouseLoc)
```

18: エクスターナル、辞書

この章の概略

- *LiveCodeのエクスターナルの説明と、エクスターナル開発キットのインフォ。*
- *ウェブ・ブラウザーの機能を作るエクスターナル「revBrowser」のスタック。*
- *メディアをリンクしたり、テキストをインポートしたりする時に必要な、開発書類が保存してあるフォルダーまでのパスを求めるファンクション。*
- *LiveCode辞書の使い方と、言葉を探すコツ。*

この章で使われる英単語

- external [名詞, 形容詞：外部, 外の]
- longitude [名詞：経度]
- latitude [名詞：緯度]
- dictionary [名詞：辞書]
- syntax [名詞：（言語学の）統語論]
- example [名詞：実例]
- summary [名詞, 形容詞：要約, 簡潔な]
- assign [動詞：当てがう, 任命する]

LiveCodeで言うエクスターナルは、スタンドアロンの外部からイメージやサウンド、ビデオ等をリンクして、アプリケーション内に表示させるメディアと言う意味と、もうひとつはLiveCodeエンジンでは扱えないコンピュータの機能を、LiveCodeよりも低レベルの（より機械言語に近い）言語で特定の機能だけを開発して、外部部品としてカスタム・コマンドやカスタム・ファンクションにして取り付ける事を指す場合とがあります。

LiveCodeの開発環境では、後者のエクスターナルがすでにビルトインされているものもあって、「revBrowser」「revFont」「revSpeech」「revZip」「revdb」「revxml」「revVideoGrabber」等がそれらです。ビルトインでなく、サード・パーティまたは独自で作ったエクスターナルを使うには、スタンドアロンを作る時に、これらの機能の外部ファイルを、スタック・インスペクターの「External References」で、アプリケーションに組み込むリンクをしなければいけません。上記のビルトインされているエクスターナルはその必要はなく、スタンドアロンに作られる時、自動でMac OSならアプリケーションのパケージ内に、Windowsなら「Externalフォルダー」にファイルごとリンクされます。Mac OSとWindowsではエクスターナル・ファイルの拡張子が違っていて、Macは「.bundle」、Windowsは「.dll」が付きます。

LiveCodeにビルトインされているエクスターナルから、LiveCodeのスクリプトに取り込む言葉は、カスタム・コマンド、カスタム・ファンクションのライブラリーですから、普通の日常英語のような表現を取らない事が多く、始めに「rev」が付いている言葉が多くあります。

エクスターナル開発に興味ある人の為に、RunRev社の解説ページ（英文）を書いておきます。
External Writing for the Unitiated – Part I
http://newsletters.livecode.com/november/issue13/newsletter5.php
External Writing for the Unitiated – Part 2
http://newsletters.livecode.com/november/issue14/newsletter3.php

ウェブページを取り込む「revBrowser」

エクスターナルの「revBrowser」は、LiveCodeがまだ「Runtime Revolution」と言っていた頃

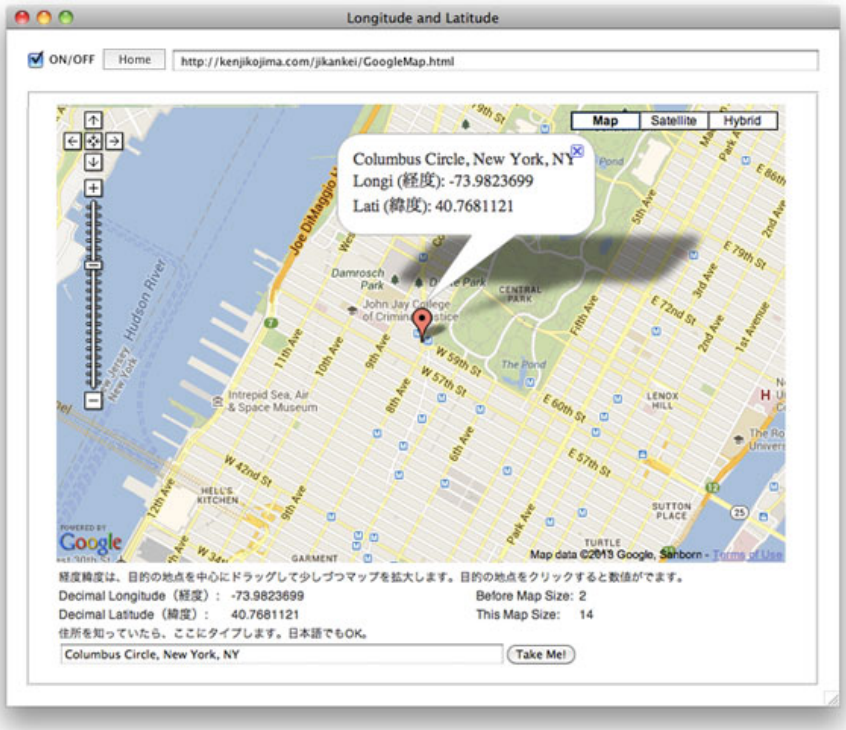
に、サードパーティの「Altuit社」が開発したエクスターナルをRunRev社が買い取り、開発キットに標準で装備されるようになりました。「revBrowser」は、スタックにウェブ・ページを取り込んでウェブ・ブラウザーの機能を作り出します。

エクスターナルの「revBrowser」を使ってみましょう。「revBrowser」は、スタックのカード上に座標で設定した四角形に、インターネット・ブラウザーの表示機能を作ります。Mac OSではSafariのWebKit、Windowsでは Internet Explorerをベースに開発されています。表示するウェブ・ページにもよりますが、SafariとInternet Explorerでは、同じHtml原稿でも天地の長さが変わってくるので、スクロール・バーを持たせないで特定のページだけを表示させたい場合、プラットフォーム（platform）別に、設定するエリアを少し変える必要があります。これからサンプルで作るスタックは汎用性を持たせて、一般的なウェブ・ブラウザーのように、ウインドウのサイズを変えたり、アドレスがユーザーに書き込めるようにしてあります。

手っ取り早く実物を見てみます。下のスクリプトを、メッセージ・ボックスにコピペしてリターンキーを叩くと、サーバーからスタック「longiLati.livecode」が呼び出せます。

```
go to url "http://kenjikojima.com/livecode/download/longiLati.livecode"
```

こんな感じのものです。Mac OSでピッタリ入るように作っているので、Windowsで見ると右にスクロールバーが出て来ます。『preOpenStack』でWindowsで開いたサイズも調整できますが、今回はそこまでやっていません。私がLiveCodeで作った「時間系不定時報」と言う、地球上各地の太陽の運行を表示させるアプリケーションの、補助のウェブ・ページです。Google Mapの機能から緯度経度を知るウェブ・ページを、このチュートリアル用に日本語にして整理しました。少し試してみてから作ってみましょう。



ニュー・メインスタック（New Mainstack）を作って名前を「longiLati」、タイトルを「Longitude and Latitude」として、ファイル「longiLati.livecode」を保存してください。開かれ

た時に「http://kenjikojima.com/jikankei/GoogleMap.html」を表示させるようにします。スタックのサイズは左右（width）「800」、天地（height）「650」にします（上図）。左上にチェック・ボックス、ボタン名は「onOff」レイベルは「ON/OFF」を置いて、その右にホームに戻るボタン「Home」、その右にアドレスのためのフィールド（Text Entry Field）の名前は「tUrl」にして、「Don't Wrap（dontWrap）」にします。その下にブラウザーの表示エリアをグラフィック（graphic）で作り、名前は「tBrowser」にします。図では薄いグレーの矩形の部分です。回りのアキは、スタック・サイズが変わっても、それぞれ20ピクセル取るようにスクリプトを書きます。ブラウザー機能の表示には、プレイヤーやイメージのようにオブジェクトの必要はないのですが、レクタングル（rectangle）の範囲がはっきり分かるように、グラフィック「tBrowser」をカード上に置きます。

```
revBrowserOpen(ウインドウID, アドレス)
```

このファンクションでブラウザーを初期化をして、セッションのイニシャルIDを取得します。IDはブラウザーを表示させるプロパティをコマンドの

```
revBrowserSet イニシャルID, プロパティ名, プロパティの値
```

のパラメータに使います。プロパティは「ボーダーを見せる」「表示範囲」の２つを設定します。これがカードに入れる大雑把なスクリプトの流れです。

カードに入れるスクリプトをを書きます。コマンド「revBrowserOn」が中心的なスクリプトです。エクスターナル「revBrowser」を使う機会があったら、これをそっくりそのままコピーして使うのが良いかもしれません。他のプロパティをセットするパラメータは、辞書の「revBrowserSet」にあります。この章に「辞書を使う」の項目があります。スクリプトから日本語は削除してください。

```
----------  カード内のスクリプトここから ----------
-- オブジェクト内すべてのハンドラーで使えるようにローカルを宣言
local lBrowserId

on openCard
   if btn "onOff" is not hilite then
      hilite btn "onOff"
   end if
   revBrowserOn
end openCard

command revBrowserOn
  -- スタックのwindowIDを「tWinID」に入れる
  put the windowID of this stack into tWinID

  -- フィールドにあるウェブ・アドレスを「tUrl」に入れる
  put fld "tUrl" into tUrl

  -- レブ・ブラウザーを初期化してIDを「lBrowserId」に
  put revBrowserOpen(tWinID,tUrl) into lBrowserId

  -- レブ・ブラウザーのプロパティ（ボーダー、表示範囲）を設定
```

```
  revBrowserSet lBrowserId, "showborder","true"
  revBrowserSet lBrowserId, "rect",rect of grc "tBrowser"
end revBrowserOn

on revBrowserOff
  revBrowserClose lBrowserId
end revBrowserOff


-- ここから「resizeStack」まではウインドウのサイズが変わった時のためです
-- ジオメトリック・マネージャーを使うと
-- レブ・ブラウザーの書き換えができないので「resizeStack」を使います

-- コンスタント（constant  定数）「tSpace」を20（ピクセル）にします
constant tSpace = 20

on resizeStack x,y   -- x,y はスタック・ウインドウの左右、天地
  -- ウインドウ・サイズが変わったら、フィールドのレクト（rect）を変える
  set the rect of fld "tUrl" to tSpace *5, tSpace, x -tSpace, tSpace *2

  -- ウインドウ・サイズが変わったら、
  --   ブラウザーの表示範囲（グラフィック「tBrowser」のレクト）を変える
  set the rect of grc "tBrowser" to \
        tSpace, tSpace *3, x -tSpace, y -tSpace

  -- もしカードにあるボタンがチェックしてあったら
  --   ウインドウ・サイズが変わった時、ブラウザーを描き変える
  if the hilite of btn "onOff" then
    revBrowserSet lBrowserId, "rect", the rect of grc "tBrowser"
  end if
end resizeStack


-- URLが変わる前に許可を求める。これは必ずしも必要ではないけれど
--   「revBrowser」の機能のひとつに付け加えました。削除しても問題ありません
on browserNewUrlWindow pId, pUrl
   answer "link trying to open in new window:"&cr&pURL with "No" or "OK"
   if it is "OK" then
      put pURL into fld "tUrl"
      revBrowserSet pId, "url", pUrl
   end if
end browserNewUrlWindow
----------   カード内のスクリプトここまで ----------
```

カード上のボタン「onOff」のスクリプト

```
on mouseUp
  if hilited of me then
    revBrowserOn
  else
    revBrowserOff
  end if
```

```
end mouseUp
```

カード上のボタン「Home」のスクリプト

```
on mouseUp
   put "http://kenjikojima.com/jikankei/GoogleMap.html" into fld "tUrl"
   send returnInField to fld "tUrl"
end mouseUp
```

カード上のフィールド「tUrl」のスクリプト

```
on returnInField
  hilite btn "onOff"
  revBrowserOn
end returnInField
```

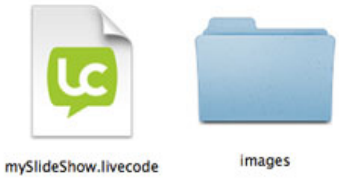
開発書類が保存してあるフォルダー

クロスプラットフォームでエクスターナルのメディアをリンクしたり、テキストをインポートしたりする時に必要なファンクションです。一通りの説明は付けましたが、細かな処がわからなくても、ファンクション「theFolderPath」として、このように使えれば問題ないです。

```
set the fileName of image "myslide" to \
        theFolderPath() & "/images/mySlide.jpg"
```

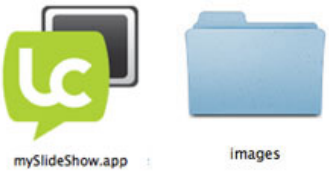
ファイル・パスを得るファンクションは、イメージ・オブジェクトのファイルネーム（fileName）などをセットする時に使います。MacOSの開発中のメインスタックと、スタンドアロンのメインスタックでは返される「the fileName of this stack」が違っているので、それを修正するファンクションです。MacOSではスタンドアロン・アプリケーションの中に「Contents」が入っていて、またその中の中にスタックがあるので、スタンドアロンと同じホルダー（階層）に置いている、他のファイルとのパスに違いが出てしまいます。スタンドアロン・アイコンの右クリックで中を見る事ができます（図：mySlideShow.app）。

開発時にデスクトップに置いたフォルダー「LiveCodeStudy」の中に、下の図のように開発書類の「mySlideShow.livecode」と、イメージを入れたフォルダー「images」があるとします。開発しているスタックのファイルネーム「the fileName of this stack」を求めると




```
/Users/kenjikojima/Desktop/liveCodeStudy/mySlideShow.livecode
```

このような結果が返されます。しかしスタンドアロンにして、スタックのファイルネーム「the fileName of this stack」を求めると




```
/Users/kenjikojima/Desktop/
            liveCodeStudy/mySlideShow.app/Contents/MacOS/mySlideShow
```

左右の文字詰めの関係で２行にしています。

が返されます。開発時とスタンドアロンでは同じ結果が得られないので、ファイルパスを求める為の修正ファンクションが必要です。ファンクション「theFolderPath」は、メインスタックのスクリプト・エディターに書き込んで使います。

```
function theFolderPath
   if the environment is "development" then
      set the itemDel to "/"
      return item 1 to -2 of the filename of this stack
   else
      put the fileName of this stack into tPath
      set the itemDel to "/"
      switch the platform
         case "MacOS"
            put item 1 to -5 of tPath into tResult
            break
         case "Win32"
            put item 1 to -2 of tPath into tResult
            break
      end switch
      return tResult
   end if
end theFolderPath
```

```
ファンクション　theFolderPath
    もし環境が「development　開発」なら
        アイテムデリを「スラッシュ　/」に設定して
        このスタックの「filename　ファイルネーム」のアイテム１から−２を返しなさい
    そうでなかったら
        このスタックの「filename　ファイルネーム」を「tPath」に入れなさい
        アイテムデリを「スラッシュ　/」に設定して
        スイッチ文始まり　プラットフォームをチェックして行く
            ケース　「MacOS」なら
                「tPath」のアイテム１から−５までを「tResult」に入れる
                ブレイク（スイッチを抜ける）
            ケース　「Win32」なら
                「tPath」のアイテム１から−２までを「tResult」に入れる
                ブレイク（スイッチを抜ける）
        スイッチ文終わり
        「tResult」を返す
    if文終わり
```

```
theFolderPath 終わり
```

赤文字にした「this stack」はメインスタックだけの場合は正しいフォルダー・パスを返しますが、サブスタックでも使う場合サブスタックを「this stack」と捉えるので、メインスタック、サブスタック共通で使えるように、「this stack」はメインスタックの名前に変えてください。例えば

```
return item 1 to -2 of the filename of stack "mySlideShow"
```

のようにします。

```
case "MacOS"
     put item 1 to -5 of tPath into tResult
```

スタンドアロンのMacOSでは、アプリケーションのパケージに「Contents」と言うフォルダーからのパスが、メインスタックまで続いているので、その分を削除します。

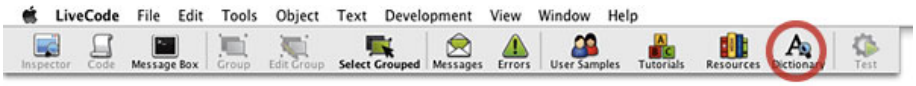
スタンドアロンと同じフォルダーにある「images」フォルダー内の「slide01.jpg」をイメージ・オブジェクト「mySlide」にセットするには

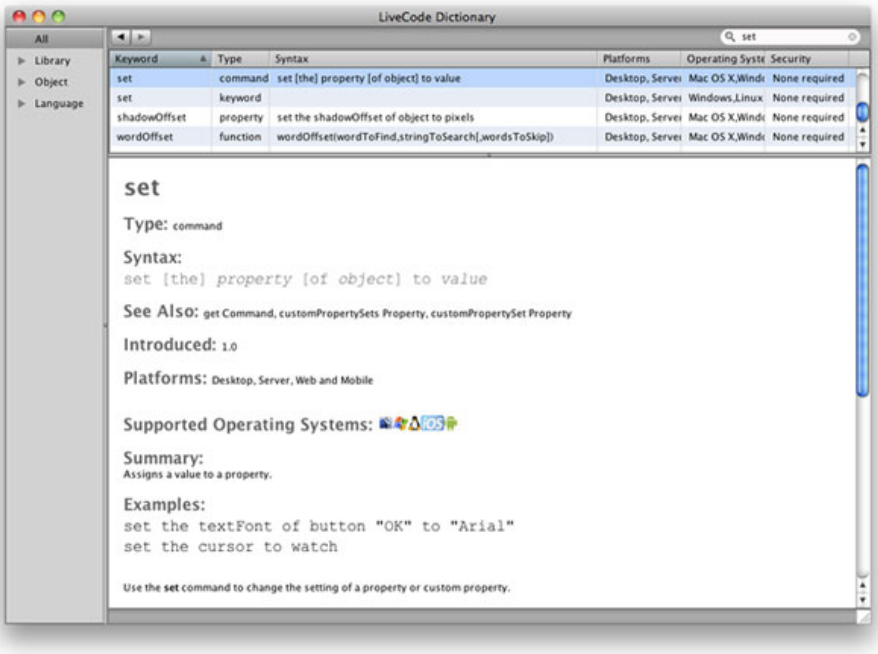
```
set the fileName of image "mySlide" to \
        theFolderPath() & "/images/slide01.jpg"
```

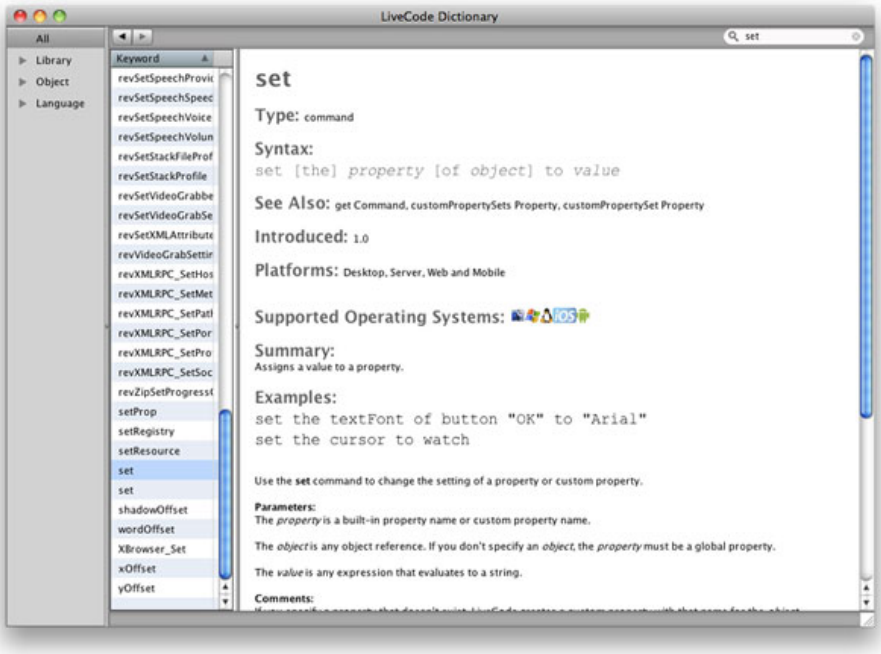
のように使います。

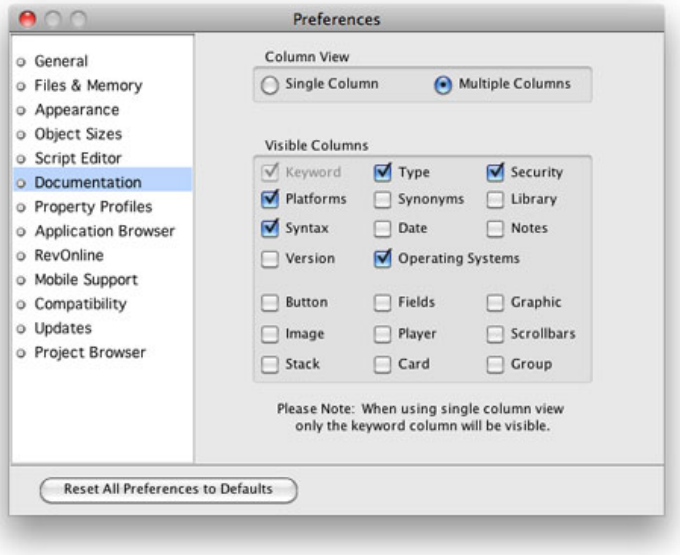
辞書を使う

LiveCodeのめぼしいテーマは一通りなぞって来ました。これからLiveCodeの開発をして行く過程で、頻繁に辞書(Dictionary) を使うことになるでしょう。アイコン・メニューの右にある「A」をクリックすると、辞書（LiveCode Dictionary）が開かれます。




LiveCode Dictionary
All
Library
Object
Language
set
Keyword
Type
Syntax
Platforms
Operating Syste
Security
set
command
set [the] property [of object] to value
Desktop, Server
Mac OS X,Windo
None required
set
keyword
Desktop, Server
Windows,Linux
None required
shadowOffset
property
set the shadowOffset of object to pixels
Desktop, Server
Mac OS X,Windo
None required
wordOffset
function
wordOffset(wordToFind,stringToSearch[,wordsToSkip])
Desktop, Server
Mac OS X,Windo
None required
set
Type: command
Syntax:
set [the] property [of object] to value
See Also: get Command, customPropertySets Property, customPropertySet Property
Introduced: 1.0
Platforms: Desktop, Server, Web and Mobile
Supported Operating Systems:
Summary:
Assigns a value to a property.
Examples:
set the textFont of button "OK" to "Arial"
set the cursor to watch
Use the set command to change the setting of a property or custom property.


LiveCode Dictionary
All
Library
Object
Language
set
Keyword
revSetSpeechProvic
revSetSpeechSpeec
revSetSpeechVoice
revSetSpeechVolun
revSetStackFileProf
revSetStackProfile
revSetVideoGrabbe
revSetVideoGrabSe
revSetXMLAttribute
revVideoGrabSettir
revXMLRPC_SetHos
revXMLRPC_SetMet
revXMLRPC_SetPatl
revXMLRPC_SetPor
revXMLRPC_SetPro
revXMLRPC_SetSoc
revZipSetProgressI
setProp
setRegistry
setResource
set
set
shadowOffset
wordOffset
XBrowser_Set
xOffset
yOffset
set
Type: command
Syntax:
set [the] property [of object] to value
See Also: get Command, customPropertySets Property, customPropertySet Property
Introduced: 1.0
Platforms: Desktop, Server, Web and Mobile
Supported Operating Systems:
Summary:
Assigns a value to a property.
Examples:
set the textFont of button "OK" to "Arial"
set the cursor to watch
Use the set command to change the setting of a property or custom property.
Parameters:
The property is a built-in property name or custom property name.
The object is any object reference. If you don't specify an object, the property must be a global property.
The value is any expression that evaluates to a string.
Comments:



辞書は、マルチ・コラムとシングル・コラムの２種類から選ぶ事ができます（上図１、２）。マルチ・コラムはトップが「keyword」「Type」「Platforms」「Operating Systems」「Security」のコラムが表示されて、その下に辞書の本文があります。シングル・コラムは左に「keyword」だけが表示されて、その右に辞書の本文があります。ふたつの表示の違いは、LiveCodeのメニューから引き出せるPreferences(環境設定）の「Documentation」から設定できます（上図３）。もうひとつLiveCodeのウェブ・ページにもありますから、どれでもあなたの好みで使ってください。

API (Language Dictionary): http://livecode.com/developers/api/6.0.2/

引き出した辞書の内容は、詳しく読むに越した事はないですが、そう英語も読みたくはないでしょうから、見方のポイントだけを説明しておきます。

辞書の左側のコラムから「All」を選んで、右上検索フィールドに例えば「set」をタイプすると、「set」が含まれるすべてのLiveCodeの言葉がリストに表れます。setはかなりの語が現れますが、普通はこれほど沢山はリストアップされません。リストをスクロールダウンして行くと、「Keyword」の欄に「command」と「keyword」の２つの「set」があるのがわかります。今はコマンドの「set」を見てください。リストのラインがハイライトになってその内容が出ます。

まずトップに選んだ語の「set」があって、次の「Type: command」は、この言葉がどのタイプに属しているかを示しています。他に「function」「property」「keyword」「message」などに分類されます。

その下の「Syntax　シンタックス」は、日本語辞書だと「統語論」「統辞法」などに訳されています。意味不明なので「シンタックス」と私は言っています。LiveCode的な意味でシンタックスを日本語にすると、基本文型とでも言いますか、文の成り立ちに必要な単語や要素を組み合わせた、基本的な文の型と言う意味です。シンタックスはこの辞書の中でも大事な部分で、読み方さえ分かれば理解しやすいですから、必ず見てください。

```
set [the] property [of object] to value
```

このコマンド「set」のシンタックスを見ると、スクエア・ブラケット「 [ ] 」に囲まれた語とイタリックの語があります。シンタックスは１つだけとは限らないで、何行か書かれている場合もあります。シンタックスのスクエア・ブラケットに入った語、この場合例えば「 [the] 」は、オプションの言葉なので、場合によって省くことができます。たぶんこの辺りの判断は、英語的な感覚で「the」を付けた方が、読んだ時に語感の落ち着きが良いとかなんでしょう。他の言葉で「the」を使う場合は注意してください。ファンクションでパラメータなしで「the」を必ず付ける事があります。

それからイタリックで書かれた部分、たとえば「set」の例だったら「property」や「value」は、他の言葉や数値に、置き換える部分です。具体的な置き換える語の例は、下に書いてある「Examples（イグザンプル　例文）」にあります。

別の語「put」を見てみます。

```
put value [{before | into | after} container]
put value into URL destinationURL
```

カーリー・ブラケット「{}」に囲まれた語は、文の構成には不可欠な語で、文を成り立たせるためには、どれかひとつを選ぶ必要があります。選ぶ語はバーティカルバー「| 」(タテ棒、パイプとも言う)によって区切られています。「put」のこの例では、スクエア・ブラケットの中にカーリー・ブラケットがありますから、もしオプションとしてスクエア・ブラケット内のコンテナー(container) を使ったら、カーリー・ブラケット内の一つの語は文を構成するために必ず必要です。

「Syntax」の下の「See Also：」には、その語と関連のある他の語が列記されていますから、関係のありそうな語をクリックして、参考に見る事ができます。

その下の「Summary（サマリー　要約）」は、言葉の意味、または役割を簡単に記述したものです。「set」にある「Assigns a value to a property.」は「値をプロパティに割り当てる(設定する)」と言う意味になりますが、サマリーは無理に日本語の文章に翻訳しないで、単語の意味だけ調べてシンタックスと例文から類推した方が、理解できるかもしれません。

Summary（サマリー）の下の「Examples（イグザンプル　例文）」は、シンタックスと同じに大事な部分です。シンタックスの語順通りのスクリプトが例としてあげられていますから、シンタックスと対で見てください。例文は数行ある事も多く、どう使われるのかを具体的に知ることができます。

```
set the textFont of button "OK" to "Arial"
```
ボタン「OK」のテキストフォントを「Arial」にセットしなさい。

```
set the cursor to watch
```
カーソルをウオッチにセットしなさい。

辞書から目的の語を探し出す

LiveCodeの言語は何度か書いているように、日常英語に非常に近いので、プログラム上で何かしたいことがあって、言葉がわからない時は英語で考えてみて、その事に当てはまる日常的な英語の言葉を辞書で検索窓に打ち込んで、LiveCodeの言葉をリストア ップしてみてください。例えばユーザーが色を選べる「カラー・パレット」を出したと思ったら、「color」と辞書で検索してみます。リストアップされた中から、だいたいの当りをつけて「answer color」のようだと思ったら、本文を見れば分かります。外れても少しは記憶に残って、また何かの時に思い出すでしょう。日本語から和英辞書を使っても参考になるでしょう。また出てきた言葉の説明を見て、それをヒントにまた次の言葉を探し出して、目的のLiveCodeの言葉にたどり着くなどということも良くあります。辞書の「See Also：」にある言葉も、とてもに参考になります。またLiveCodeは非常に柔軟性のある表現が可能ですから、例文などにはない言い回しで使えることも、しばしばあります。探し出した言葉はすぐにランタイムで試す事ができるので、思いついた可能性はとにかく試してみることが大事です。細かなことはすべて覚えきれないので、基本的な事柄を頼りに辞書をうまく使いこなして行くのが、上達の早道でしょう。

Tips

- ビルトイン・エクスターナルはスタンドアロンにした時、自動でリンクされて「Externals フォルダー」にファイルが入れられます。

- 開発環境とスタンドアロンのファイルパスの違いは、ファンクション「theFolderPath」で調整します。

- ファンクション「theFolderPath」で、サブスタックからファンクション「theFolderPath」を使う時は、「this stack」は「stack "スタック名"」に変更しないと、正しいパスが得られません。「theFolderPath」は「empty」を返します。

- 適当なLiveCodeの言葉が思い当たらない時は、知っている意味の近い単語を辞書で検索してみます。時には日本語から和英で単語を探し出すのも有効かもしれません。ランタイムでとにかく試してみる事が大事です。

この章で新しく出て来た言葉

revBrowserOpen
ファンクション（function）レブ・ブラウザーをオープンして初期化する
```
put revBrowserOpen(tWinID,tUrl) into lBrowserId
```

revBrowserSet
コマンド（command）レブ・ブラウザーのプロパティを設定する
```
revBrowserSet lBrowserId, "showborder","true"
revBrowserSet lBrowserId, "rect",rect of grc "tBrowser"
```

revBrowserClose
コマンド（command）レブ・ブラウザーをメモリーから削除する
```
revBrowserClose lBrowserId
```

constant
コマンド（command）定数を宣言する

```
constant tSpace = 20
```

browserNewUrlWindow
メッセージ（message）レブ・ブラウザーが閉じられる時に送る

```
on browserNewUrlWindow pId, pUrl
  answer "I will open "&cr& pUrl
end browserNewUrlWindow
```

environment
ファンクション（function）スタックが現在走っている環境

```
put the environment
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

19: サンプル・スタックから、次のステップに進む

この章の概略

- *ここから先に進むための参考になるサンプル・スタック*
- *スライドショーのサンプル・スタックと、日本語のコメントをスクリプトに書き込むプラグイン「jpComments」*

これで第一部「ほとんど英語テキストでの開発」は最終章です。LiveCodeの開発で押さえておくべき事柄は一通り書いたので、次のステップに進むには、開発しようと思ったり、興味のある分野のサンプル・スタックを見る事が大事です。メニューには「User Samples」「Tutorials」「Resources」と３種類に分類したサンプル・スタックがあります。たぶん英語を読むのはメンドウと言う人も多いでしょう。英語の解説部分はハショっても、サンプル・スタックだけ直接動きを見ながらスクリプトを読んでも、内容はかなり分かるし、それはそれで楽しいです。

もう一カ所良質のサンプル・スタックがダウンロードできる所を書いておきます。
Tactile Media: http://www.tactilemedia.com/
http://www.tactilemedia.com/site_files/software/tutorials.html

日本語を使う開発に興味のある方は、次の第二部「日本語テキストを使う開発」を読んでください。LiveCode（RunRev）は日本語ができないと言うウワサが、誇大に広まっているようですが、何度か書いているように、オペレーション・システムでのファイルパスと、スクリプト・エディターに日本語コメントが書けないと言う問題以外は、たぶんほとんどLiveCodeの日本語の使い方を知らないで、言っているのではと思っています。日本語コメントについては、このチュートリアルのために作った、プラグインを使う簡易的な方法をこの章で書きます。

スライドショー　サンプル・スタック

このスライドショー・スタックは、ニューヨークに住んでいる私の友人が、去年東京の某大学で、開拓時代から現代までのアメリカ美術史の、ゲスト講師に招かれて講義すると言う時に、相談を受けて作ったものです。彼は15年程前まで、日本でアメリカ美術の講座を持っていましたが、現在70代後半、コンピューターを使ったプレゼンテーションの経験もなく、とにかくクリックだけですべてできるように作ったものに、スクリプトに日本語コメントを付けて少し手を加えました。アシストしてくれる人に、事前にインターネットで送っておいたので、特に問題もなく講座は無事に終わったそうです。

サンプル　ダウンロード： slideshow.zip
http://kenjikojima.com/livecode/download/slideshow.zip
「slideshow.zip」を拡大すると、下の図のように「imagesフォルダー」「jpComments.livecode」「slideshow.livecode」が入っています。「imagesフォルダー」には

サンプルのJPEGファイルが５点と、イメージそれぞれに付ける日本語の説明が５行ある「text.txt」が入っています。サンプルに付けたJPEGはすべて同じサイズですが、どんな比率の写真でもOKです。日本語の説明は必要がなければ、空白にします。日本語原稿はユニコードのUTF8で保存しています。

「slideshow.livecode」が本体のスタックです。

日本語コメントを書くプラグイン

「jpComments.livecode」はこのチュートリアルのために作った、日本語のコメントをスクリプト・エディターに書き込んで、またそれを読むプラグインです。日本語のコメントを書き込むと言いましたが、実際は日本語を「base64」と言う文字コードに変換して、LiveCodeのスクリプト・エディターにコピー＆ペーストします。それをもう一度プラグインの「jpComments.livecode」で読み込んで、プラグインのフィールドに日本語で表示します。私は今まで英語でコメントを書き込んで来たので、特に必要ではありませんが、日本で少しでもLiveCodeが使いやすいように、どうしても日本語でコメントを書き込みたい人のために作ってみました。オープン・ソースですから、だれか低レベルの言語でLiveCode本体の改善をしてくれれば一番良いですが、それまでの簡易的な日本語化です。

LiveCodeのソースはここからダウンロードできます。
https://github.com/runrev

それとユニコード（日本語）は扱いさえ分かっていれば、これくらいの事はできると言う事と、開発ツールはLiveCodeの言葉で自作できるデモでもあります。

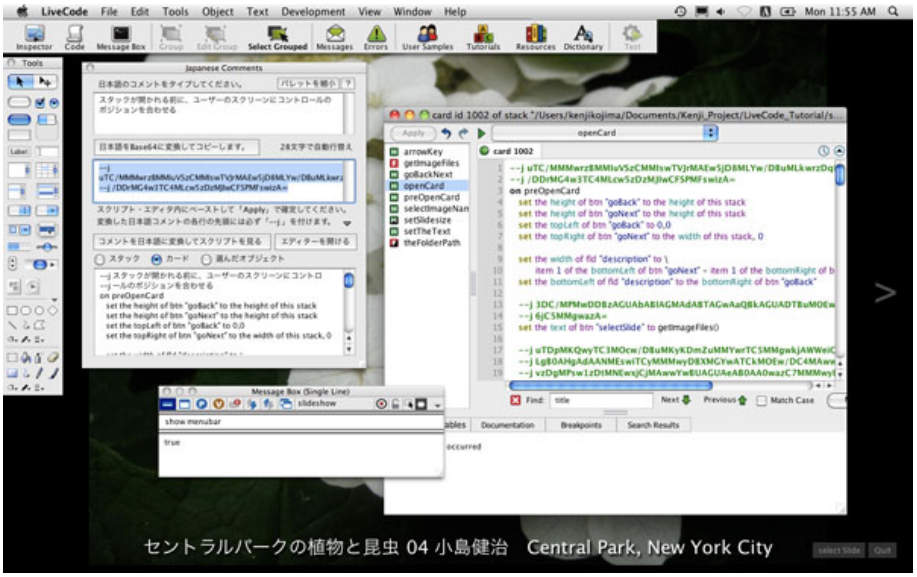
スライドショーはフルスクリーンなので、スタックを開くとLiveCodeの開発ツールもメニューも見えなくなります。まず始めに「jpComments.livecode」から開いてください。常時使うようでしたら、プラグイン・フォルダーにインストールするのが良いです。今は「Fileメニュー」からオープンします。

トップにあるフィールドに日本語のコメントをタイプして、その下「日本語をBase64に変換してコピーします。」をクリックすると、その下のフィールドに始めに「--j 」の付いた英数字が表れて、その時点で変換されたその文字列がクリップボードにコピーされます。左右は自動で28文字に行替えされます。行が変わった始めには必ず「--j 」が付いています。開発中のスタックのスクリプト・エディターに変換されたその文字列をペーストしたら、スクリプトを確定させるスクリプト・エディター左上の「Apply」をクリックするか、MacOSなら「コマンド・キー + リターン・キー」。Windowsは「コントロール・キー ＋ リターン・キー」を叩いてください。確定させないと「コメントを日本語に変換してスクリプトを見る」は正しく働きません。長いスクリプトをプラグインのテキスト・フィールドで見るのが大変な時は、スクリプト・エディターから「--j 」のラインだけをコピーして、プラグインの２番目のフィールドにペーストしたら、ラジオ・ボタン「上のフィールド」を選んで「コメントを日本語に変換してスクリプトを見る」をクリックします。

「slideshow.livecode」を開く前に、メッセージ・ボックスを開いてください。スタックを開いたら

```
show menubar

hide this stack
```

どちらか一行をメッセージ・ボックスから送ります。Windowsでは「hide this stack」だけが有効です。サンプル・スタックなので、特に編集の必要はないのですが念のため。
「slideshow.livecode」を開くと、スクリーン全面にスタックが開かれます。下の図はメニューやスクリプト・エディターも開いています。スクリプト・エディターは「jpComments」からも開けるようにしています。



「jpComments」の下半分で、最上部にあるスタックのスクリプトを、Base64から日本語に変換して見る事ができます。

```
hide menubar

show this stack
```

で、フルスクリーンのスタックに戻す事ができます。

スライドショー・スタックのスクリプト全文

以下に日本語のコメントを付けたスクリプトを書きます。左右の字詰めの関係で、実際のスクリプトとは違う行替えの箇所があります。

---------- 「slideshow」スタック内のスクリプトここから ----------

```
--// this stack was programmed by Kenji Kojima, index@kenjikojima.com //----

--j スタンドアロンにする前に必要な事柄を初期化します
command setStandalone
   set the filename of image "slide" to empty
   put empty into fld "description"
   set the cText of image "slide" to empty
end setStandalone
```

---------- 「slideshow」スタック内のスクリプト終わり ----------

スタンドアロンにするプロジェクトには、だいたいいつもコマンド「setStandalone」を作っておきます。フィールド内の文字や、ラジオ・ボタンのセットなどの初期化をするために、スタンドアロンのテストの前にメッセージ・ボックスから送ります。これは私のやり方ですから、また別な方法もあるでしょう。

```
---------- 「slideshow」カード内のスクリプトここから ----------
--j スタックを開く前に、メニューバーを隠して、フルスクリー
--j ン、黒のバックグラウンドに。フルスクリーンにコントロー
--j ルのポジションを合わせる
on preOpenStack
   hide menubar
   set the fullscreen of stack "slideshow" to  true
   set the backgroundColor of this stack to 0,0,0
   set the height of btn "goBack" to the height of this stack
   set the height of btn "goNext" to the height of this stack
   set the topLeft of btn "goBack" to 0,0
   set the topRight of btn "goNext" to the width of this stack, 0
   set the width of fld "description" to the width of this stack
   set the bottomLeft of fld "description" to \
        the bottomLeft of  btn "goBack"
   set the bottomRight of grp "SelectQuit" to \
    item 1 of the bottomRight of this stack - 6, \
    item 2 of the bottomRight of this stack - 8

--j ボタン「selectSlide」のメニューをイメージの
--j リストに
   set the text of btn "selectSlide" to getImageFiles()

--j スライドショーの説明のテキストを外部ファイル「text
--j .txt」からゲットして、イメージ「slide」のカス
--j タム・プロパティ「cText」にセットする
   get url ("binfile:text.txt")
   if it <> empty then
      set the cText of image "slide" to it
   else
      set the cText of image "slide" to empty
   end if

   showFirstImage
end preOpenStack


private command showFirstImage
   put line 1 of getImageFiles() into tSlideName
   setTheText 1
   set the filename of image "slide" to tSlideName
   setSlidesize
end showFirstImage


--j Macの開発環境とスタンドアロンの「filename」
--j の違いを修正するファンクション
private function theFolderPath
   if the environment is "development" then
      set the itemDel to "/"
      return item 1 to -2 of the filename of this stack
   else
      put the fileName of this stack into tPath
```

```
      set the itemDel to "/"
      switch the platform
         case "MacOS"
            put item 1 to -5 of tPath into tResult
            break
         case "Win32"
            put item 1 to -2 of tPath into tResult
            break
      end switch
      return tResult
   end if
end theFolderPath


function getImageFiles
   put theFolderPath() & "/images" into tFolder
   set the defaultFolder to tFolder
   put the files into tFiles

--j フォルダー「images」内のJPEGファイル以外をフ
--j ィルターで取り除く
   filter tFiles with "*.jpg"
   return tFiles
end getImageFiles

--j 矢印キーが叩かれた時にスライドを変える
on arrowKey tKey
   if tKey is "right" then
      send mouseUp to btn "goNext"
   else
      if tKey is "left" then
         send mouseUp to btn "goBack"
      end if
   end if
end arrowKey


--j スライド説明の文（日本語、英語）をフィールド「desc
--j ription」にセットする
command setTheText pLineNum
   if the cText of image "slide" is not empty then
      set the unicodeText of fld "description" to \
        uniencode(line pLineNum of the cText of image "slide", utf8)
   end if
end setTheText


--j 左右に配置した「goNext」「goBack」
command goBackNext
   put the filename of image "slide" into slideName
--j イメージのファイルネームから、説明テキストの何行目かを
--j 「tImageNum」に入れる
   put lineOffset(slideName, the text of btn "selectSlide") \
```

```
     into tImageNum

   if word 2 of the target is quote& "goNext" "e then
      if tImageNum >= the num of lines of getImageFiles() then
--j 始めに戻る
         put line 1 of getImageFiles() into tSlideName
         setTheText 1  --//fld "description" //---
      else
         put line tImageNum +1 of getImageFiles() into tSlideName
         setTheText tImageNum +1  --//fld "description" //---
      end if
   else
      if tImageNum is 1 then
--j 現在始めのイメージを見せているので、goBackで最後
--j のイメージにする
         put line -1 of getImageFiles() into tSlideName
         setTheText the num of lines of the cText of image "slide"
           --//fld "description" //---
      else
         put line tImageNum -1 of getImageFiles() into tSlideName
         setTheText tImageNum +1  --//fld "description" //---
      end if
   end if
   set the filename of image "slide" to tSlideName
   setSlidesize
end goBackNext


--j イメージをセットして、説明文をフィールド「descri
--j ption」にセット
command selectImageName pItemName
   set the filename of image "slide" to pItemName
   put lineOffset(pItemName, getImageFiles() ) into tLineNum
   set the unicodeText of fld "description" to \
      uniencode(line tLineNum of the cText of image "slide", utf8)
   setSlidesize
end selectImageName


--j イメージを左右又は天地の最大サイズにセット
private command setSlidesize
   put the formattedwidth of image "slide" into tFormatWidth
   put the formattedHeight of image "slide" into tFormatHeight
   put the width of this stack into tStackWidth
   put the height of this stack into tStackHeight
   if tFormatHeight >= tFormatWidth then
      set the height of image "slide" to tStackHeight
      set the width of image "slide" to tFormatWidth * \
        tStackHeight div tFormatHeight
   else
      put tStackWidth - 2 * the width of btn "goNext" into tSlideWidth
      set the width of image "slide" to  tSlideWidth
      set the height of image "slide" to tFormatHeight * \
```

```
        tSlideWidth div tFormatWidth
   end if
   set the loc of image "slide" to the loc of this stack
end setSlidesize
```
---------- 「slideshow」カード内のスクリプト終わり ----------

---------- 「slideshow」ボタン「goNext」スクリプトここから ----------
```
on mouseUp
   goBackNext
end mouseUp
```
---------- 「slideshow」ボタン「goNext」スクリプト終わり ----------

---------- 「slideshow」ボタン「goBack」スクリプトここから----------
```
on mouseUp
   goBackNext
end mouseUp
```
---------- 「slideshow」ボタン「goBack」スクリプト終わり ----------

---------- 「slideshow」ボタン「selectSlide」スクリプトここから ----------
```
on menuPick pItemName
   selectImagename pItemName
end menuPick
```
---------- 「slideshow」ボタン「selectSlide」スクリプト終わり ----------

---------- 「slideshow」ボタン「Quit」スクリプトここから ----------
```
on mouseUp
   quit
end mouseUp
```
---------- 「slideshow」ボタン「Quit」スクリプト終わり ----------

3D ワイヤーフレーム ドローイング

最後にアニメーション・エンジンを使ったスタックを紹介します。私のプロジェクトの「RGB MusicLab」と言うアプリケーションのサブスタックの１つです。始めにアニメーション・エンジンを開いておかないと、エラーになります。アニメーション・エンジンは「13章：マルチメディア１」で説明しています。まだ持っていなかったら、ダウンロードはこのアドレス
http://forums.runrev.com/viewtopic.php?f=27&t=14399&sid=4e6165c8465c6fd1282381e35d1aa83b
からできます。サイトに行ってアニメーション・エンジンのライセンスを読んで、「animationEngine5.0.2.zip」をダウンロードしてアニメーション・エンジンを開いたら、「Use me!」をチェックにします。それからメッセージ・ボックスで
```
go stack url "http://kenjikojima.com/livecode/download/draw3D.livecode"
```
を、送ってください。

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

ショートカット

	Mac OS	Windows
ブラウズ・ツールを選ぶ	Command-9	Control-9
エディット・ツールを選ぶ	Command-0	Conttrol-0
パレットを出す隠す	Command-Control-Tab	Control-Tab
コンテクスト・メニューを出す	Command-Control- Shift-Click	Control-Shift-右Click
すべてのスタックを保存	Command-Option-s	Control-Alt-s
カード１を開ける	Command-1	Control-1
前のカードを開ける	Command-2	Control-2
次のカードを開ける	Command-3	Control-3
最後のカードを開ける	Command-4	Control-4
すべてを選択	Command-a	Control-a
複製を作る	Command-d	Control-d
新しくカードを作る	Command-n	Control-n
１ピクセル移動	Arrow キー	Arrow キー
10ピクセル移動	Shift-Arrowキー	Shift-Arrowキー
スタック・インスペクターを開く	Command-k	Command-k
ペイント・ツールで正方形を描く	Shift	Shift
オブジェクトを正方形にする	Shift	Shift

スクリプト・エディター関連

オブジェクトをセレクトして	Command-e	Control-e
カードのエディター	Command-Shift-c	Control-Shift-c
スタックのエディター	Command-Shift-s	Control-Shift-s
オブジェクトをマウスで	Command-Option-Click	Control-Alt-Click
スクリプトの変更を確定	Enter	Enter
スクリプトの変更を確定して ウインドウを閉じる	Enterを２回	Enterを２回
スクリプトの変更を確定して 保存する	Command-s	Control-s
ラインをコメントにする	Command-hyphen	Control-hyphen
ラインのコメントを取る	Command-Shift-hyphen	Control-Shift-hyphen
ファインド（検索）	Command-f	Control-f

メッセージ・ボックス関連

開く／閉じる	Command-m	Control-m
以前のメッセージ（シングルライン）	Arrowキー	Arrowキー
以前のメッセージ（マルチライン）	Option-Arrowキー	Alt-Arrowキー
実行（シングルライン）	Return	Return
実行（マルチライン）	Enter-Control-Return	Enter-Control-Return

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

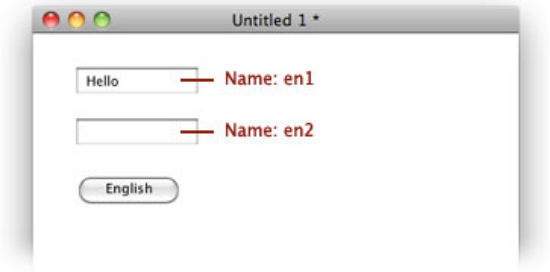
20：日本語（ユニコード）の基礎

この章の概略

- *フィールド間で日本語テキストを移動させる。*
- *ボタンのレイベルやスタックのタイトルに日本語を設定します。*
- *フォントの違いから言語を分類して、フィールドの日本語をフォント別に設定します。*
- *インターネットにある日本語をテキスト・フィールドにインポート。*
- *他のユニコード・フォーマットから、LiveCodeで使うユニコードに変換。*

フィールドからフィールドに文字列を移動

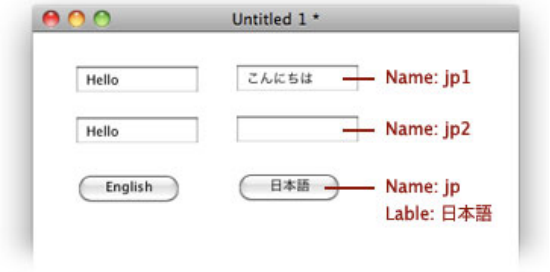
始めに英語でやってみます。ニュー・メインスタックを作って、テキスト・フィールドを２つツール・パレットから作ります。上に置いたフィールドの名前（name）は「en1」、下のフィールドは「en2」にします。フィールド「en1」に「Hello」とタイプして



「English」と言うレイベル（lable）のボタンを作って、中のスクリプトは

```
on mouseUp
   put fld "en1" into fld "en2"
end mouseUp
```

として、ボタンをクリックするとフィールド「en2」に「Hello」と文字列が入ります。それでは日本語をやってみます。同じようにフィールドを２つ、名前を「jp1」と「jp2」にします。ボタンは名前（name）を「jp」、レイベル（label）を「日本語」にしてください。



これで英語と同じように

```
on mouseUp
   put fld "jp1" into fld "jp2"
end mouseUp
```

と送ると「?????」がフィールドに入ります。日本語では「put ... into 」は使えません。 今「日本語」と書いていますが、LiveCodeでは日本語だけでなく、普通の英数字以外の文字を使う国の言葉は、すべてUnicode（ユニコード）と言う文字コードを使います。別な言い方をすると、ユニコードに対応したスクリプトにしておけば、中国語でも、韓国語でも、ロシア語でも、何語でも対応可能と言う事になります。とは言っても、私たちのよく知らない歴史的な経過で、何種類かのユニコードというのがあって、LiveCodeのユニコードの表示は「UTF-16」と言うユニコードを使います。インターネットでは「UTF-8」と言うユニコードがポピュラーなでの、「UTF-8」から「UTF-16」の変換や、その逆の「UTF-16」から「UTF-8」の変換も、この第二部で扱います。もうひとつ日本語では「SJIS」と言う文字コードもポピュラーですから、その変換についても書いて行きます（SJISはユニコードではありません）。

英語のようにASCIIの256文字以内で扱える文字コードを「シングルバイト言語（１バイト）」と言い、それに対して日本語のよう言語を「ダブルバイト言語（２バイト）」と言っています。LiveCodeのスクリプトは、「シングルバイト」でのみ書く事ができます。LiveCodeのスクリプトは、ダブルバイト言語とミックスするとトラブルになりますから、それを避ける方法についても書いて行きます。

ASCIIについては10章、12章に説明があります。

「put」はフィールド、「set」はそれ以外

それでは続きをやりましょう。バージョン5.5からLiveCodeに加えられた言い回しの

```
put unicode ユニコードにした文字列 into fld フィールド名
```

から説明します。これは文章の取り扱いに「put」を使う、ハイパートークからの流れに沿った言い回しで、それ以前のLiveCodeのユニコードの扱いは、すべて「set」を使っていました。「put unicode 」を使う場合は、フィールドの文字列に限られます。それ以外のオブジェクトに日本語（ユニコード）を表示させたい場合は「set」を使います。今まで使っていた「set」は、引き続きフィールドでも使う事はできます。現段階では「put」を使うのはフィールド、その他は「set」と分けて考えた方が良いと思われます。「set」を使う方法は、バイナリーのイメージ・データを、イメージ・オブジェクトにセットしたのと少し似ています。実際ファイルを保存する場合など、LiveCodeではユニコードをバイナリーデータとして取り扱います。

「put」を使う具体的な方法について書きます。フィールド内にある文字列を、「ユニコードにした文字列」とするのには「the unicodeText of fld フィールド名」を使います。

```
put unicode the unicodeText of fld "jp1" into fld "jp2"
```
フィールド「jp1」のユニコード・テキストをフィールド「jp2」にプット（put）しなさい

これで「put」を使って、日本語文字を他のフィールドに移し替える事ができます。

では「set」を使ってやってみます。日本語の入っているフィールドも、これから日本語を入れるフィールドも「the unicodeText of fld フィールド名」にして、ターゲットのフィールドに「set」します。

```
set the unicodeText of fld "jp2" to the unicodeText of fld "jp1"
```
ユニコードテキストのフィールド「jp2」をユニコードテキストのフィールド「jp1」に設定しなさい。

これでこちらも、フィールド「jp2」に「こんにちは」が入ります。
フィールド「jp1」をユニコードにしないで、普通のテキストの意味「fld "jp1"」にすると

```
set the unicodeText of fld "jp2" to fld "jp1"
```

何だか分からない漢字の文字化けになってしまいます。英語に比べると「put」も「set」も、ちょっとだけ面倒ですね。とにかくこれがユニコードを扱う最も基本の形です。どちらもユニコードにした文字列で扱わないと、きちんと表示されません。「put」はユニコード・テキストの文字列をフィールドにプット（put）します。「set」はユニコード・テキストの文字列を、ユニコード・テキストを受けられる形にしたオブジェクト（フィールドに限りません）に、セット（set）すると覚えてください。

ボタンのレイベルをスクリプトで日本語に

上に書いたユニコードを扱う基本の形のバリエーションで、ボタン名「jp」のレイベルを日本語表記にスクリプトでしてみます。今度はメッセージ・ボックスから送ってください。

```
set the unicodeLabel of btn "jp" to the unicodeText of fld "jp1"
```
ボタン名「jp」のユニコードのレイベルを、ユニコードテキストのフィールド「jp1」に設定しなさい。

これでボタン「jp」のレイベルも「こんにちは」になりました。ほとんどの場合「the unicodeLabel of btn ボタン名」を使わないで、英語と同じように「the label of btn ボタン名」でも、日本語レイベルをセットできますが、「上」や「伊」などのユニコードUTF16の中の一部の文字が、トラブルを起こすことがあって、日本語のボタン表示をする場合は、「the unicodeLable of」を使ってください。ユニコードUTF16の中の一部の文字のトラブルは、後の章で説明する予定です。

ユニコードを扱っていると、ちょっとしたスペル・ミスや文法の間違いで、文字化けが起こる事があります。場合によっては、LiveCodeがフリーズしたりするかもしれません。フリーズしたら、Macなら、アップル・メニューから「強制終了...」で、Windowsなら「タスク・マネージャー」でLiveCodeを終了させてください。

ランゲージからフォントを分類

ユニコードを扱うマナーに従わない設定が、文字化けを引き起こして場合によったらフリーズする事もありますが、他の原因に適切なフォントに設定されていないと言う場合もあります。スクローリング・フィールド（scrolling field）をカード上に作って、メッセージボックスから

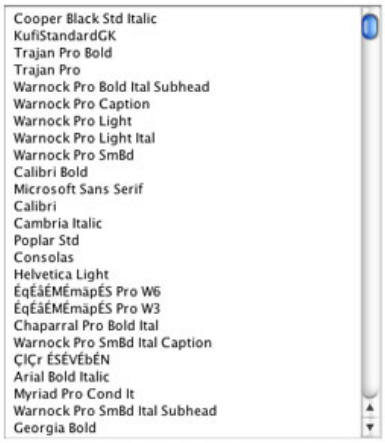
```
put the fontNames into fld 1
```

を送ると、すべてのフォントのリストがフィールドに入れられます。しかしいろいろな処で文字化けを起こしているのが見られます。「put unicode ユニコードの文字列 into fld フィールド名」で、ユニコードに変換してやってみます。しかし フィールドに入っている文字列なら「the unicodeText of fld "フィールド名」と言う形でユニコードを取り出せますが、「the fontNames」のような文字列をユニコードにするには、「uniEncode(文字列)」を使います。それではこれで、メッセージボックスから送ってみます。「uniEncode」を使わないと、フリーズする可能性がありますから注意！

```
put unicode uniencode(the fontNames) into fld 1
```

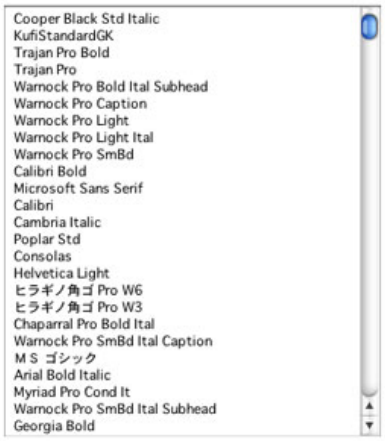
これでも、やっぱり同じように文字化けしています。では「set」を使ってみます。

```
set the unicodeText of fld 1 to uniEncode(the fontNames)
```



一応マナー通りの設定をしたのですが、まだ文字化けしています。このザ・フォント・ネーム（the fontNames）はいくつもの言語の文字が入っているので、それに見合ったフォントに設定しないと、英語の文字しか正しく表示されません。MacOSでは「Lucinda Grande」が、Windowsでは「Segoe UI」がデフォルトでフィールドのフォントに、デフォルトで設定されているので、MacOSだったら「Osaka」に、Windowsだったら「Tahoma」をテキスト・フィールドに設定すれば、英語と日本語が読める文字になります。

```
set the texFont Of fld 1 to "Osaka"   -- Mac OS

set the texFont Of fld 1 to "Tahoma"   -- Windows
```



これほぼ文字化けはなくなりましたが、まだいくつか文字化けの行があります（日本語、英語フォント以外入れてない場合は、文字化けはありません）。このフォント・ネーム（the fontNames）で得られるも文字列は、それぞれの行がランゲージのインフォメーションを含んでいるので、日本語や英語は「Osaka（MacOS）」や「Tahoma（Windows）」では文字化けは起こさないでも、韓国語や、タイ語や、トルコ語などがリストに入っているとその行は文字化けになります。

```
put the fontLanguage of fld 1
```

と、メッセージ・ボックスから送ると、MacOS（Osaka）もWindows（Tahoma）も「Japanese」を返します。日本語か英語を使っている人達のコンピュータでは、フォント・ネーム（fontNames）は「ANSI(English fonts)」か「Japanese」を返しますが、他のランゲージが入っていると、それぞれ別なランゲージ名を返します。ほとんど使う事もない言語もあるかもしれませんが、「the fontLanguage」は以下の言語が返されます。と言う事は、以下の言語をLiveCodeではユニコードで扱う事ができます。

ANSI (English と同じに使われます)
Arabic
Bulgarian
Chinese
Greek
Hebrew
Japanese
Korean
Polish
Roman
Russian
Thai
Turkish
SimpleChinese
Ukrainian

スタックに日本語タイトル。
日本語フォントでテキストをセットする

フォント・ランゲージ（fontLanguage）で日本語フォントだけを選び出して、日本語テキストを、それぞれのフォントにセットしてみます。作っているスタックを「fontLanguage.livecode」と言うファイル名で保存してください。日本語でスタックのタイトルを付けます。スタックの何処かにフィールドを作って名前（name）を「jpTilte」としてメッセージ・ボックスから

```
set the unicodeTitle of this stack to the unicodeText of fld "jpTitle"
```

このスタックのユニコード・タイトルを、
フィールド「jpTitle」のユニコードテキストにセットしなさい。

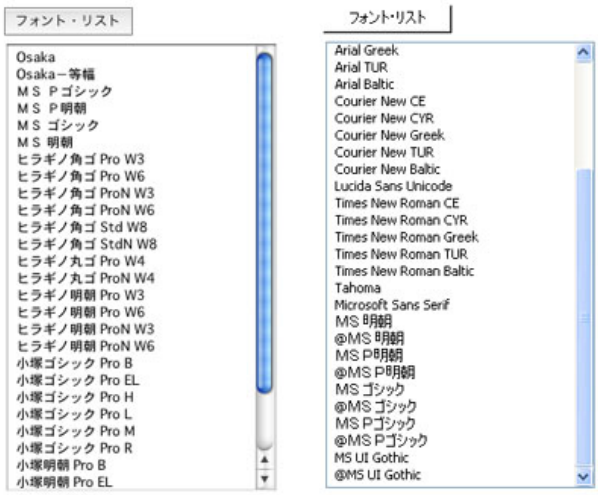
ここで日本語タイトルのために使ったフィールド「jpTilte」は、もう使わないので、削除しても良いです。オブジェクトに日本語のレイベルやタイトルを付ける時に、メッセージ・ボックスから行うと上手く行かない事が多いので、こうしてフィールドを補助的に使う事が頻繁に出てきます。

日本語のフォント名だけのリストを作ります。作ってあるスクローリング・フィールドの名前を「tFontNames」としてください。ボタンを作ってレイベルを「フォント・リスト」にしました。「the fontNames」を使ってこのマシーンにインストールされているすべてのフォント名をリストアップして、さらに「the fontLanguage」が「Japanese」のもののみを書き出します。MacOSの場合はソートした方が奇麗に整理がつくので、ソートしてからフィールドのテキスト・フォント（textFont）を「Osaka」に、Windowsはソートしない方が奇麗に整理されているので、ソートしないでテキスト・フォント（textFont）を「Tahoma」にします。

```
on mouseUp
   set the cursor to watch
   get the fontNames
   repeat for each line tLine in it
      if the fontLanguage of tLine is "Japanese" then
         put tLine &cr after tJpFonts
      end if
   end repeat
   if the platform is "MacOS" then
      sort tJpFonts
      set the textFont of fld "tFontNames" to "Osaka"
   else
      set the textFont of fld "tFontNames" to "Tahoma"
   end if
   delete char -1 of  tJpFonts
```

```
   set the unicodeText of fld "tFontNames" to uniencode(tJpFonts)
end mouseUp
```

```
オン　マウスアップ
    カーソルをウオッチにセット
    フォント名のリストをゲットする（itに収納される）
    リピート文 it の中のそれぞれのラインを「tLine」にして繰り返す
        もしフォント・ランゲージが「Japanese」なら
            「tLine」と行替え（cr）を「tJpFonts」の後ろにつける
        もし（if）終わり
    リピート終わり
    もしプラットフォームが「MacOS」なら
        「tJpFonts」をソートしなさい
        フィールド「tFontNames」のテキスト・フォントを「Osaka」に設定
    （プラットフォームがMacOS）でなかったら
        フィールド「tFontNames」のテキスト・フォントを「Tahoma」に設定
    もし（if）終わり
    「tJpFonts」の最後のキャラクターを削除しなさい
    ユニコードにした「tJpFonts」を
          ユニコードテキストのフィールド「tFontNames」にセットしなさい
エンド　マウスアップ
```



MacOSでは上の左図、Windowsでは右図のようになります。Windowsでは英語名でも「Japanese」を返すフォントが多くあります。これらのフォント名で日本語テキストをセットします。始めにそれぞれのフォント名がどんな書体なのか試してみましょう。ボタンを作ってレイベルを「フォント名にセット」にしました。

```
on mouseUp
   set cursor to watch
   lock screen
   get uniDecode(the unicodeText of fld "tFontNames",english)
   repeat with i=1 to the num of lines of it
      set the textFont of line i of fld "tFontNames" to line i of it
   end repeat
end mouseUp
```

```
オン　マウスアップ
    カーソルをウオッチにセット
```

```
    スクリーンをロック
    フィールド「tFontNames」のテキストをゲットして「it」に入れる
    リピート文　「i」を1からitのラインの数にして繰り返す
        フィールド「tFontNames」のライン「i」のテキスト・フォントを　\
            「it」のライン「i」にセットしなさい
    リピート文終わり
エンド　マウスアップ
```

フォント名をそれぞれのフォントにセットしました。

始めはユニ・エンコード「uniEncode」を使って、フォント名のリストをユニコードのフィールドにセットできるようにしました。「フォント名にセット」ボタンでは、フォント名をユニコードにしていると、スクリプトの中で使えないので、直接フィールドからテキストをゲットして、シングルバイト文字で返されたフォント名をそのまま使っています。ユニコードのフィールドからユニコードとして取り出して、それをユニデコード（uniDecode）して、「ANSI（English）」にした事と同じ内容です。

```
get the text of fld "tFontNames"
  直接ユニコードのフィールドから文字列をゲット

get uniDecode(the unicodeText of fld "tFontNames",english)
  ユニコードのフィールドから、一旦ユニコードを取り出して、　\
        ユニデコード（uniDecode）を使って、ANSII（english）に変換
```

上の２行はまったく同じ働きをします。下の行はいったんユニコードで取り出して、スクリプトで使える「English」に変換しています。

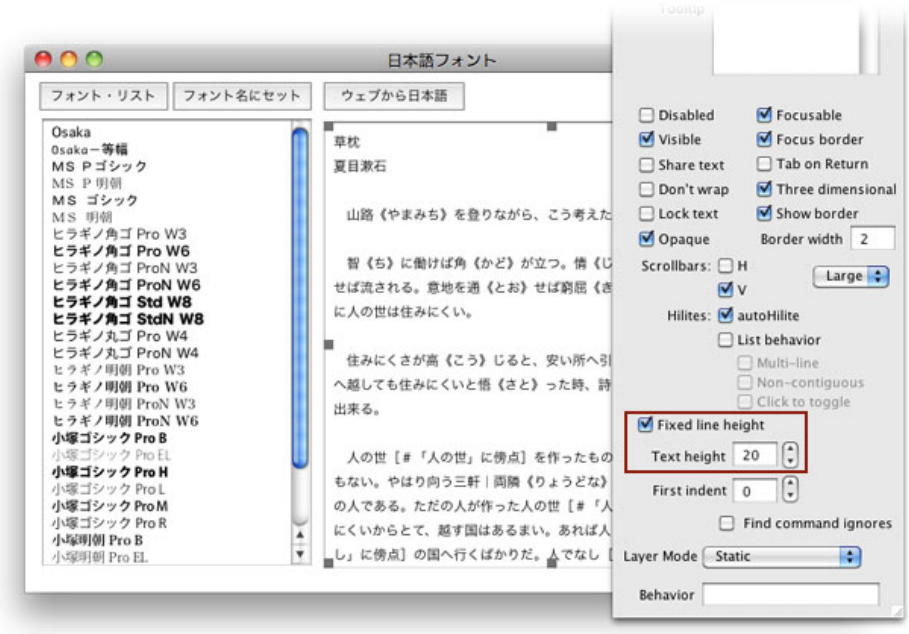
次に日本文のサンプル・フィールドを作って、日本語文を入れます。ウェブに夏目漱石の草枕の始めの部分を、ユニコードの「UTF8」で作ったテキスト・ファイルが置いてありますから、カード上のテキスト・フィールドに取り込みます。フォント名のリストを入れたフィールド「tFontNames」の右隣に、スクローリング・フィールド「jpText」を作ってください。ボタン「ウェブから日本語」を作ってスクリプトは、

```
on mouseUp
   get url "http://kenjikojima.com/livecode/download/kusamakura.txt"
```

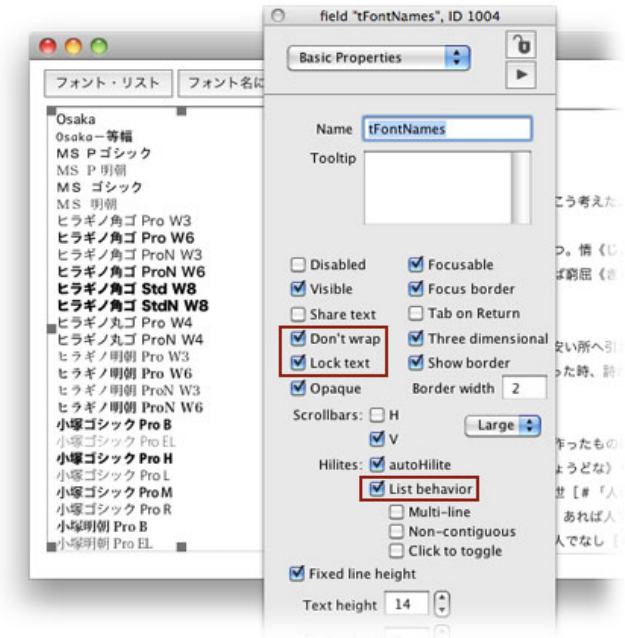
```
   set the unicodeText of fld "jpText" to uniEncode(it,utf8)
   set the textHeight of fld "jpText" to 20
end mouseUp
```

```
オン　マウスアップ
   URl「url "http://kenjikojima.com/livecode/download/kusamakura.txt"」をゲット
   ユニコードにしたエンコードしたUTF8の「it」を
        ユニコードテキストのフィールド「jpText」にセットしなさい
   フィールド「jpText」のテキストハイト（textHeight）を20にセットしなさい
エンド　マウスアップ
```

クリックしてフィールド「jpText」に日本語を取り込みます。フィールド「jpText」はデフォルトのセッティングなので、フォント・サイズが11ポイントで、行間が14ポイントになっています。日本語では行間をほぼ全角アキくらい取った方が読みやすいので、ラインハイト（textHeight）を20ポイントにしました（下図赤枠）。



次にフォント名のリストをマウスでクリックすると、フィールド「jpText」のフォントがセレクトされたフォントに設定されるようにします。フィールド「tFontNames」をロックして、クリックした行を選べるように、「Don't warp」「Lock text」「List behavior」にチェックをします（下図）。



リストが選べる事を確認したら、クリックライン（clickLine）が使えるので、フィールド「tFontNames」に下のスクリプトを入れます。

```
on mouseUp
   put the text of the clickLine into tFontName
   set the textFont of fld "jpText" to tFontName
end mouseUp
```

```
オン　マウスアップ
    クリックされたラインのテキストを「tFontName」に入れなさい
    フィールド「jpText」のフォント（textFont）を「tFontName」に設定しなさい
エンド　マウスアップ
```

フィールドをクリックすると「the text of the clickLine」で、ユニコードではないシングルバイトの『ANSI（english）』が返されますから、直接フォント名としてスクリプトで使えます。フォント名のクリックで、サンプルの日本語のフォントがセットされるようになっていますか。

日本語（ユニコード）の基礎　まとめ

- フィールドから取り出す日本語は必ず「the unicodeText of フィールド名」、フィールドにセットする日本語も必ず「the unicodeText of フィールド名」を使う。

- バリアブル（variable）に入れた日本語文字列（シングルバイト）を、日本語としてオブジェクトにセットする時は「uniEncode(バリアブル)」で、ユニコード（ダブルバイト）に変換してからセットする。

- 「UTF8」など他のテキスト・フォーマットをLiveCodeに表示するときは、「uniEncode(バリアブル,元のフォーマット名)」を使って「UTF16」に変換してからオブジェクトにセットする。

- フォント名など、ユニコードにして取り出した文字列はスクリプトでは使えないので、直接シングルバイトの「ANSI」で取り出すか、「uniDecode(ユニコードのバリアブル,english)」で、ユニコードからデコードの変換をして使う。

- ユニデコードはLiveCodeのユニコード「UTF16」から、「UTF8」や『SJIS』など、他の文字コードに変換する時にも使える。

この章で新しく出て来た言葉

unicodeText
プロパティ（property）フィールドにユニコード（ダブルバイト・キャラクター）を表示させる
```
set the unicodeText of fld "jp2" to the unicodeText of fld "jp1"
```

put unicode
コマンド（command）フィールドにユニコード（ダブルバイト・キャラクター）を入れる
```
put unicode the unicodeText of fld "jp1" into fld "jp2"
```

unicodeLabel
プロパティ（property）オブジェクトのレイベルにUTF16を表示させる
```
set the unicodeLabel of btn "jp" to the unicodeText of fld "jp1"
```

fontNames
ファンクション（function）インストールしてあるフォント名を返す
```
put the fontNames into fld 1
```

fontLanguage
ファンクション（function）ユニコード・フォントの言語を返す
```
put the fontLanguage of fld 1
```

uniEncode
ファンクション（function）シングバイト・キャラクターをユニコードに変換する
```
set the unicodeText of fld 1 to uniEncode(the fontNames)
```

uniDecode
ファンクション（function）ユニコードをシングバイト・キャラクターに変換する
```
get uniDecode(the unicodeText of fld "tFontNames",english)
```

unicodeTitle
プロパティ（property）スタック・ウインドウのタイトルにユニコードを表示する
```
set the unicodeTitle of this stack to the unicodeText of fld "jpTitle"
```

textHeight
プロパティ（property）フィールドにあるテキストの行間のスペース
```
set the textHeight of fld "jpText" to 20
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

21：フィールドにある日本語の扱い

この章の概略

- *日本語テキストを、ラインやキャラクター単位で移動*
- *日本語テキストを、UFT8、SJISのプラットフォーム共通ファイルにして保存*
- *日本語テキストを、バイトオーダーマークを付けたUTF16ファイルで保存*
- *コードの違う日本語ファイルを、LiveCodeのフィールドにインポート*
- *ユニコードの各文字に付けられた、16進数の表記について*

日本語のラインやキャラクターの移動

ニュー・メインスタックに、スクロール・フィールドを上下に２つ作って、上のフィールドの名前（name）は「jp1」、下のフィールドは「jp2」にしてください。文字列のライン・ナンバーが分かりやすいように、番号を振ったテキストをウェブに用意しました。芥川龍之介の薮の中の一部を「UTF8」で保存したテキスト・ファイルです。フィールド「jp1」にインポートしてください。「Import Text」と言うボタンに

```
on mouseUp
   get url "http://kenjikojima.com/livecode/download/yabunonaka.txt"
   put unicode uniEncode(it,utf8) into fld "jp1"
end mouseUp
```

スタックやフィールドのサイズは適当に広げてください。ライン09は空白です。念のため何行あるかメッセージ・ボックスから確かめてみます。

```
put the num of lines of fld "jp1"
```

「13」と返されます。前章で行ったフィールド内の全ての文章を、他のフィールドに移すのはこのスクリプトでした。

```
put unicode the unicodeText of fld "jp1" into fld "jp2"
```

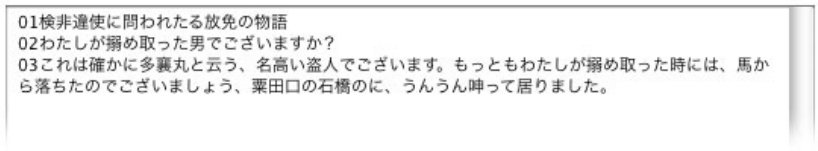
これから文章の部分的な移し替えをやってみます。始めにライン1から2までを、フィールド2に移します。メッセージ・ボックスからでも、テンポのボタンを作って中に書いてでも良いです。ハンドラーの「mouseUp」は省略します。

```
put unicode the unicodeText of line 1 to 2 of fld "jp1" into fld "jp2"
```

ほとんどは、前章のすべてのテキストを移すスクリプトと同じですが、「line 1 to 2 of 」を加えて移すラインの範囲を明記しました。

次に今２行のラインが入っているフィールド「jp2」の後ろに、ライン３だけを置きます。

```
put unicode uniencode(cr) & \
        the unicodeText of line 3 of fld "jp1" after fld "jp2"
```



ライン３だけを取り出してテキストの最後に置くと、行替えがなされませんから、ユニエンコードした行替え「uniencode(cr)」を始めに付けて、フィールドの文の後ろ（after）に置きます。こうして改行も続けて１行に書く場合、改行（cr または return）やタブ（tab）等の記号は、必ずユニコード（UTF16）にしないと文字化けしますから注意してください。

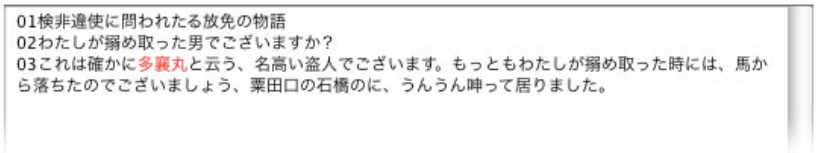
ただし改行（cr または return）だけを、ユニコードのフィールドにスクリプトでプット（put）すると、フィールドは改行（cr または return）をユニコードとして扱います。この下の２行で書いたスクリプトは、上の１行で書いたスクリプトと同じ動作をします。

```
put cr after fld "jp2"
put unicode the unicodeText of line 3 of fld "jp1" after fld "jp2"
```

どういう事で、こういう仕様になっているのか些細な事ですが注意が必要です。

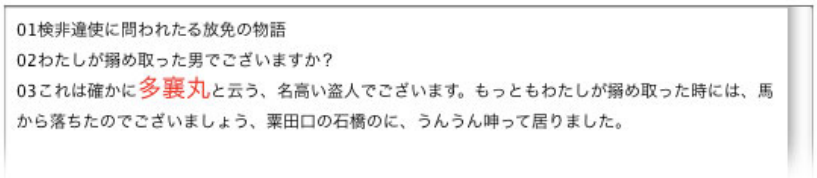
特定のキャラクターを指定して、テキスト・カラーを変えてみます。ライン3の9番目から11番目までのキャラクター「多襄丸」を赤文字にします。

```
set the textColor of char 9 to 11 of line 3 of fld "jp2" to red
```



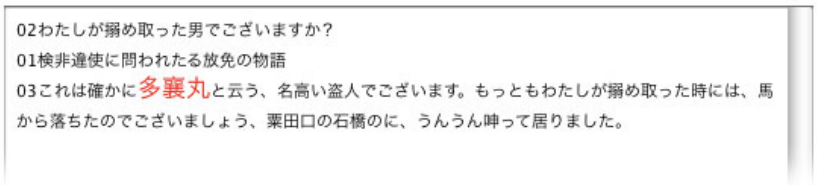
この辺りは英語と同じ扱いです。テキスト・サイズやテキスト・ハイトも英語と同じに扱えます。ライン3の9番目から11番目までのキャラクター「多襄丸」を16ポイントにして、フィールド全体のテキスト・ハイト（行間）を20ポイントにします。

```
set the textSize of char 9 to 11 of line 3 of fld "jp2" to 16
set the textHeight of fld "jp2" to 20
```



カット＆ペーストも、英語と同じようにできます。ライン1（01）をカットして、ライン2（02）とライン3（03）の間にペーストします。

```
cut line 1 of fld "jp2"
select before line 2 of fld "jp2"
paste
```



ライン1（01）をカットすると言う事は、最後に付いている改行（cr または return）も一緒にカットしていますから、ペーストする位置に改行を入れる必要はありません。「select before line 2 of fld "jp2"」で、ポジションにカーソルを入れて、ペーストします。ライン1（01）はカットされてない状態ですから、ここで使っている「line 2 of fld "jp2"」は「03」の事です。

日本語をクロスプラットフォームで保存する
始めはUFT8です

何度も書いていますが、LiveCodeの日本語（ユニコード）は「UTF16」で表示します。始めに「UTF8」で保存する方法を書きます。次に「SJIS」です。「UTF16」は少し面倒な事柄を含んでいるので、最後に説明します。

英語のテキストは「url ("file:" &ファイル・パス) 」を使って保存しましたが、ユニコードはイメージと同じように、バイナリー・ファイルで保存します。基本形は以下の通りです。

```
put uniDecode(the unicodeText of fld フィールド名, utf8) \
         into url ("binfile:" & ファイル・パス)
```

基本はフィールドにある文字列をユニコードで取り出したら、ユニ・デコード（uniDecode）で「UTF8」に変換して、ファイル・パスにバイナリー（binfile）で保存します。しかしMacOSとWindowsでは改行の形式が違っているので、それを差し替えて保存しないと、Windowsのテキスト編集アプリで開いた時に、改行されない１行に繋がった文章になってしまいます。MacOSでは改行は、「cr」または「return」を使います。どちらも「ASCII 10」ですが、Windowsの改行は「crlf（ASCII 13）」を使います。

フィールド「jp1」の日本語を「UTF8」で保存します。「Cancel」された時のスクリプトは省略しています。

```
ask file "Save UTF8 File:" with "jpUtf8.txt"
put the unicodeText of fld "jp1" into jpUTF16
replace uniEncode(cr) with uniEncode(crlf) in jpUTF16
put uniDecode(jpUTF16, utf8) into jpUtf8
put jpUtf8 into URL ("binfile:" & it)
```

１）ファイルを保存する場所を得る「ask file」で、ファイル・パスを「it」に入れます。
２）フィールド「jp1」をユニコードで取り出して「jpUTF16」に入れます。
３）「jpUTF16」のすべてのユニコードの改行「cr（ASCII 10）」を、ユニコードの「crlf（ASCII 13）」に差し替えます。
４）バリアブル「jpUTF16」に入っている日本語文（UTF16）を、「uniDecode」で「ユニコードUTF8」に変換して、バリアブル「jpUtf8」に入れます。
５）バリアブル「jpUtf8」を、URL「binfile:」で「it」に入れてあるパスの場所のファイルに入

れます。

これでMac OSのテキストエディットでも、Windowsのノートパッドでも開く事ができます。

日本語をSJISで保存する

最近はユニコードで保存する事がかなり一般的になっていますが、SJISで保存される事もまだかなりあるようです。LiveCodeではSJISを「Japanese」として扱います。

```
ask file "Save SJIS File:" with "jpSJIS.txt"
put the unicodeText of fld "jp1" into jpUTF16
replace uniencode(cr) with uniencode(crlf) in jpUTF16
put uniDecode(jpUTF16, japanese) into jpSJIS
put jpSJIS into URL ("binfile:" & it)
```

ほとんど「UTF8」と同じで、ユニ・デコード（uniDecode）を「UTF8」とするか、赤文字で書いた「Japanese」とするかの違いです。

「SJIS」は、単独のフラットフォーム用にファイル保存する場合は、バイナリーでなく「URL ("file:" & ファイル・パス)」で保存する事もできます。しかしファイルがどちらでも使えるように、バイナリーを使った方が良いでしょうね。

日本語のUTF8とSJISをインポートする

ファイルの保存は、ユニコード16を「UTF8」でユニ・デコード（uniDecode）しましたから、「UTF8」をLiveCodeにインポートするのは、バイナリーで受けたデータ（tUrl）を「UTF8」でユニ・エンコード（uniEncode）をします。「put "" into fld "jp2"」は、同じ内容のファイルをインポートするので、ファイルが入れ替わっている事を確認する為で、通常は必要ではありません。

```
put "" into fld "jp2"    -- 通常は必要ありません
answer file "Select UTF8 File:"
put url ("binfile:" & it) into tUrl
put unicode uniEncode(tUrl, utf8) into fld 2
```

「SJIS」の場合も全く同じで、ユニ・エンコード（uniEncode）を「SJIS（Japanese）」で行うだけの違いです。

```
put "" into fld "jp2"    -- 通常は必要ありません
answer file "Select UTF8 File:"
put url ("binfile:" & it) into tUrl
put unicode uniEncode(tUrl, japanese) into fld 2
```

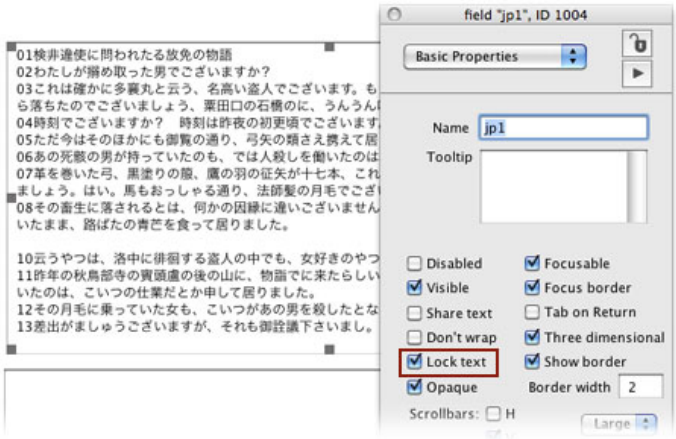
BOMを付けたユニコードUTF16の保存

LiveCodeだけに限った事ではありませんが、「UTF16」を扱うにはちょっと単純でない問題があって、その事を少しお話してからにしましょう。

まずユニコードのファイルを作る場合、ファイルの先頭に「Byte Order Mark (BOM)」と言われるキャラクターコードを付けると言うのが、ユニコードの規格を決めている「ユニコードコンソーシアム」で決められています。これによって、プロセッサーの違い(MacOSかWindowsか)を判断できるようにします。と言っても、現在ではMacとWindowsのプロセッサーが同じになっているので、古いPowerPC用に作るファイルでなければ違いはなくなっているのですが、バイトオーダー・マーク（BOM）を付けると言う事だけは、そのままに残っています。

ユニコードは１文字がダブルバイト・キャラクターですから、普通の英数字（シングルバイト＝8ビット）を２文字合わせたメモリーで１文字（16ビット）を表示します。プロセッサーによってユニコード１文字に使うメモリーの組み合わせ方の順序が違っていて、その違いをビッグエンディアン、またはリトルエンディアンと言っています。つまりプロセッサーによる読み込み方の違いを、始めに「FEFF」と言う16進数で表した文字を、バイトオーダー・マーク（BOM）として付けておく事で、それをがどういう順序になっているか正しい読み込みの順序を判断させると言うものです。

すべてのUTF16の文字には16進数の数字が振り当てられていて、LiveCodeのスクリプトの中にユニコード・キャラクターを使いたい時など、その数字からユニコードの文字に変換すると言う事もできます。サンプル・テキストのキャラクターを16進数と10進数に変換してみます。
フィールド「jp1」をロック・テキスト（lockText）にしてください。



フィールド「jp1」には以下のスクリプトを入れます。

```
on mouseUp
   if clickCharChunk() is empty then exit to top
   put unicode the unicodeText of clickCharChunk() & \
         uniEncode(cr) into fld "jp2"
   set the useUnicode to true   --「charToNum」でユニコードの番号を得る
   put charToNum(the unicodeText of clickCharChunk()) into tDec
   put "UnicodeHex: "& BaseConvert(tDec,10,16) & \
         uniencode(cr) & "UnicodeDecimal: "&tDec after fld "jp2"
```

```
end mouseUp
```

フィールド「jp1」のキャラクターをクリックすると、１行目にクリックした文字、その下に16進数で表したその文字の番号、その次の行には16進数を10進数にした数字が出ます。普通の英数字は、ASCIIと同じナンバーが10進数で出ます。「set the useUnicode to true」にすると、「charToNum」はユニコードのキャラクターを10進数に、「numToChar」は10進数をユニコードの文字に変換します。

「BOM」を作るのは、16進数の「FEFF」を10進数に変換して、その数字を「set the useUnicode to true」の状態で「numToChar」を使ってキャラクターに変換して、保存するユニコード・テキストの始めに付けます。

```
set the useUnicode to true
put numToChar(baseConvert("FEFF",16,10)) into tBom
ask file "Save UTF16 File:" with "jpBomUtf16.txt"
put the unicodeText of fld "jp1" into jpUTF16
replace uniencode(cr) with uniencode(crlf) in jpUTF16
put tBom & jpUTF16 into URL ("binfile:" & it)
```

バイトオーダー・マーク（BOM）を付けないと、LiveCodeでない他のアプリで開いた場合、文字化けする事があります。しかしこれは「UTF16」で保存したファイルを他のアプリで開く場合で、LiveCodeで再び開く場合は「BOM」は付けても付けなくても、文字化けする事はありません。

インポートは、UTF16のファイルをUTF16で表示するので、極めてシンプルです。

```
put "" into fld "jp2"   -- 通常は必要ありません
answer file "Select UTF16 File:"
put url("binfile:" & it) into tUrl
put unicode tUrl into fld "jp2"
```

この章で新しく出て来た言葉

useUnicode
プロパティ（property）
numToChar, charToNum を実行させる際ダブルバイト・キャラクターとして認識させる

```
set the useUnicode to true
put charToNum(the unicodeText of clickCharChunk()) into tDec
```

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

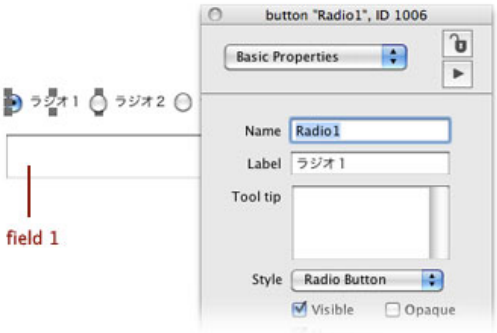
22: ボタン、メニューの日本語

この章の概略

- *日本語レイベルのラジオ・ボタンで、フィールドにそれぞれメッセージを書き出します。*
- *アラート・ウインドウを出して日本語のメッセージを表示します。*
- *選ばれた日本語のタブ・メニューから、スイッチでステートメントを引き出します。*
- *日本語のメニューを作成して、バージョン6.1の「menuPick」ハンドラーのバグを修正するスクリプトで、テキスト・フィールドのフォントをメニューから設定します。*

日本語のラジオ・ボタン

ラジオ・ボタンを３つ作って名前（name）は「Radio1」「Radio2」「Radio3」、レイベル（label）は「ラジオ１」「ラジオ２」「ラジオ３」、とします。その下にフィールド「field 1」を作ってください。



３つのラジオ・ボタンはグループにして、切り替えでハイライトになるようにします。グループ名は「radioBtn」としました。

ラジオ・ボタンはそれぞれのボタン・オブジェクトにスクリプトを書き込むのではなく、グループのスクリプト・エディターで、どのラジオ・ボタンがハイライトかを判別して指示を出します。ここでは押されたボタンの日本語レイベルをフィールドに書き込みます。どのラジオ・ボタンがハイライトになったかを知る方法には２種類あって、12章で使った「hilitedButton」は、グループ内のボタン・オブジェクトのレイヤーの数字を返します。３つのラジオ・ボタンでは「1」か「2」か「3」を返します。もう一つの「hilitedButtonName」は、ハ

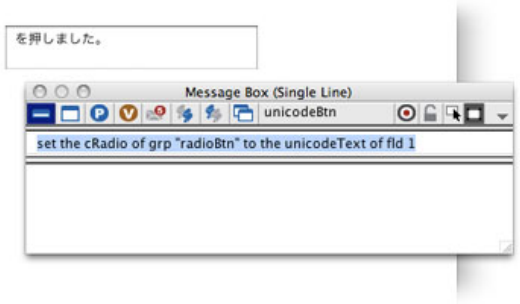
イライトされたボタン名を返します。ボタン名は英文で付けられているので、それからそのレイベルの日本語を得て、ユニコードの表示がされるようにフィールドに入れます。このスクリプトは、グループ「radioBtn」のスクリプト・エディターに書き込みます。

```
on mouseUp
   put the hilitedButtonName of me into tBtnName
   put unicode the unicodeLabel of btn tBtnName into fld 1
end mouseUp
```

「the hilitedButtonName of me」の「me」は、スクリプトを書いているラジオ・ボタンのグループの事です。これでボタン名を「tBtnName」に入れて、次の行でボタン「tBtnName」のユニコード・レイベル（日本語）を、前章のユニコードの扱いをしてフィールドに入れます。

実際のプロジェクトでは、あまりボタン名を直接書き出すと言う事も少ないでしょうから、何か違う日本文と組み合わせてフィールドに書き出す事にします。しかし直接日本語をスクリプトで扱う事はできないので、テキスト・フィールドに入っている日本語をカスタム・プロパティにしておいて、必要に応じて引き出すと言う処置で対処します。

例えばボタン１をハイライトにした時、「ボタン１を押しました」と言うメッセージがフィールドに入るようにします。「ボタン１」は上のスクリプトで取れるので、「を押しました。」と言う日本語をフィールドに入れて、それをグループ「radioBtn」のカスタム・プロパティにします。カスタム・プロパティの名前は「cRadio」としました。




```
set the cRadio of grp "radioBtn" to the unicodeText of fld 1
```

カスタム・プロパティ「cRadio」には、上の図のようなコンテンツが入ります（これはMacOSの図です）。もし開発中のスタックがクロスプラット・フォーム用ではなく、MacかWinだけの

ものでしたら、このアルファベットと記号の羅列のコンテンツを、スクリプト中にクオートで囲ったストリングとして使う事もできますが、何かの事情でクラスプラット・フォームにするかもしれないので、日本語ストリングスを扱う場合は、カスタム・プロパティにしておく事を勧めます。

ボタンが押された時点で、このカスタム・プロパティを引き出して、ボタン・レイベルの日本語名と組み合わせます。赤文字にしている部分が新たに加えたスクリプトです。

```
on mouseUp
   put the hilitedButtonName of me into tBtnName
   put unicode the unicodelabel of btn tBtnName & \
             the cRadio of me into fld 1
end mouseUp
```

日本語でアラート・ウインドウを出す

```
answer "You clicked button Radio1"
```

英語ではこのスクリプトで、ダイアログ・ウインドウを出してメッセージが表示できます。ストリングの部分を日本語にすると「???????」となってしまって、何かわからなくなります。上で作ったカスタム・プロパティを試しに使ってみます。

```
answer the cRadio of grp "radioBtn"
```

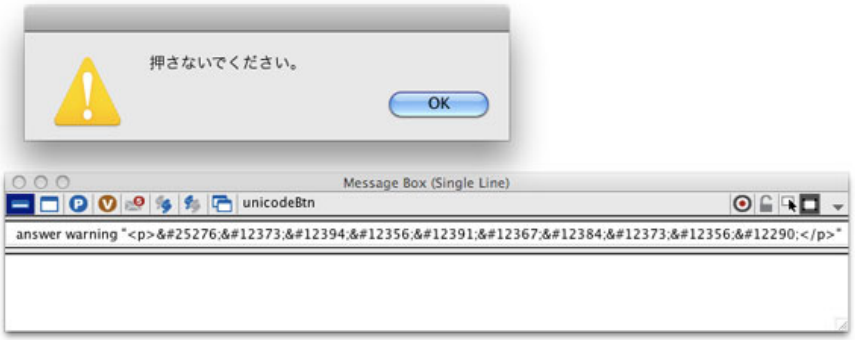
やはりこれでは日本語として読む事ができません。ダイアログに日本語を表示するのは、「htmlText」と言うのをを使います。「htmlText」はウェブ・ページを表示するプロパティで、ユニコードを「htmlText」に変換すると、キャラクター１文字づつを前章の「BOM」で説明した、UTF16に割り振った10進数のナンバーになります。グループ「radioBtn」に新しくラジオ・ボタンを追加して、名前は「radio4」、レイベルを「押さないでください。」とします。フィールド１に「押さないでください。」と日本語をタイプしてメッセージ・ボックスから

```
put the htmlText of fld 1
```

を送ると、下図のように「p」のタグで囲われた、数字が「;&#」で繋がれて返されます。ユニコードのそれぞれのキャラクターが、数字で返されたものです。

漢字「押」は「25276」「す」は「12373」です。始めにこれを使ってアラート・ダイアログを出してみます。アイコンがアラートになるように「warning」を「answer」の後に付けます。

```
answer warning "<p>&#25276;&#12373;&#12394;&#12356;" & \
          "&#12391;&#12367;&#12384;&#12373;&#12356;&#12290;</p>"
```



これで一通りの行程が理解できたと思いますから、ラジオ・ボタン「押さないでください」が押されたら、カスタク・プロパティから、このアラートを引き出すようにします。フィールドにある「押さないでください。」を「htmlText」にして、グループ「radioBtn」のカスタム・プロパティ「cRadio4」にします。

```
set the cRadio4 of grp "radioBtn" to the htmlText of fld 1
```

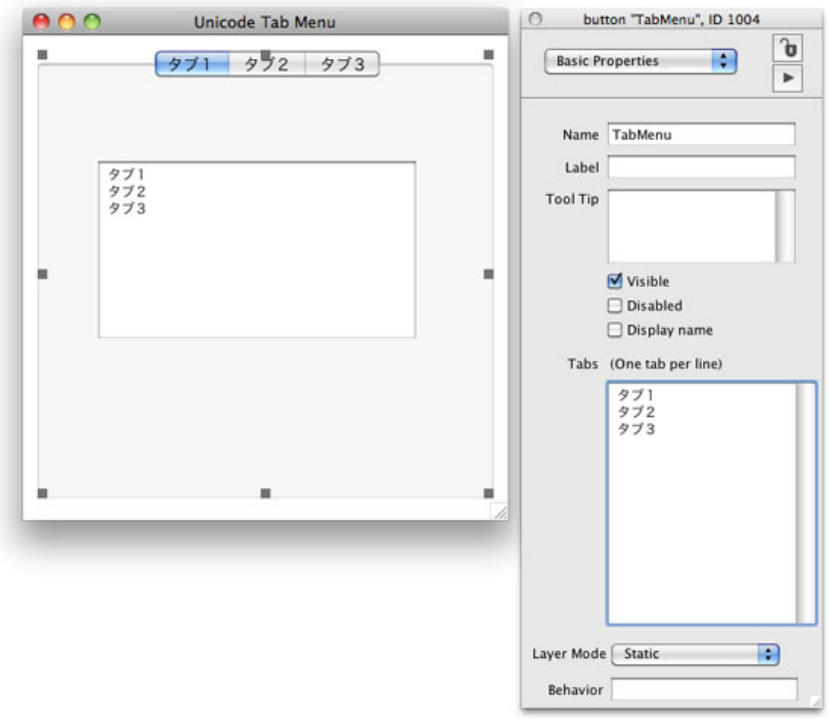
グループ「radioBtn」のスクリプトは以下のように書き換えます。ラジオ・ボタン「押さないでください（radio4）」が押されたら、ビープ音を出しアラート・ウインドウに日本語で警告して、ラジオ・ボタンのハイライトを「radio1」にします。

```
on mouseUp
   put the hilitedButtonName of me into tBtnName
   if tBtnName is "Radio4" then
      beep
      answer warning the cRadio4 of me
      set the hilitedButton of me to 1
      put "" into fld 1
   else
      put unicode the unicodeLabel of btn tBtnName & \
              the cRadio of me into fld 1
   end if
end mouseUp
```

日本語のタブ・メニュー

タブ・メニューはプロパティ・インスペクターに、直接日本語を打ち込んで日本語タブにできますが、一旦インスペクターが閉じられると、日本語タブはそのままでも、インスペクター側に打ち込んだ日本語は「?????」に変わってしまいます。しかしこのまま続けても問題ありません。もし何かの修正が必要でしたら、テキスト・フィールドに打ち込んだ日本語のタブ・リストを、メッセージ・ボックスから送ってタブにする事もできます。

```
set the unicodeText of btn "tabMenu" to the unicodeText of fld 1
```



タブ・クリックと同時に日本語タブ名を受け取って、フィールドに日本語で表示するのは、タブ・メニューに予め用意されている「menuPick」ハンドラーのパラメータ「pItemName」を使えば普通のユニコードの扱いでできます。

```
on menuPick pItemName
    put unicode pItemName into fld 1
end menuPick pItemName
```

しかし問題は、通常タブ・メニューは「switch」で選ばれたタブを「case」を使って振り分けますが、クロス・プラットフォームで統一のスクリプトにした場合、パラメータで得られるユニコードに違いが出て来るので、プラットフォーム毎に分けた「switch」コントロール・ストラクチャーを、２通り作らなくてはいけなくなります。ですから「switch」コントロール・ストラクチャーを一つだけにするには「menuHistory」を使います。「menuHistory」は、クリックされたタブ・リストの上から順に数字を返します。サンプルで言えば「タブ１」が「1」、「タブ２」が「2」、「タブ３」が「3」です。

```
on menuPick pItemName
   switch the menuHistory of me
      case "1"
         -- ここは必要に応じて、ステートメントがきます
         put unicode pItemName into fld 1
         break
      case "2"
         -- ここは必要に応じて、ステートメントがきます
         put unicode pItemName into fld 1
         break
      case "3"
         -- ここは必要に応じて、ステートメントがきます
         put unicode pItemName into fld 1
         break
   end switch
end menuPick pItemName
```

日本語のメニュー

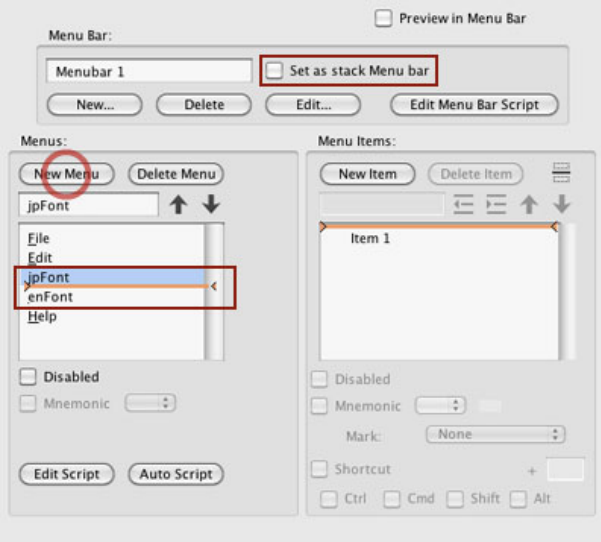
メニュー・ビルダーで作るような形の日本語のメニューです。メニュー・ビルダーのメニューは、プルダウン・メニュー・ボタンがグループ化して作られています。個々のプルダウン・ボタンのプロパティ「showBoarder」をfalseに設定すると、プルダウンされるテキストのフォントのサイズも変わって、いわゆるメニューの形体になります。現在バージョン6.1で、Windowsメニューのメニュー・ハンドラー「menuPick」が、ユニコードのメニュー・アイテムを受け取れないバグがあります。ここで書くのはそのバグを避けて選んだユニコードのメニュー・アイテムを得る、MacOSとWindows共通のスクリプトにしています。ユニコード関連のバグはなかなか修正されない事も多いので、このバグが何時修正されるか不明です。修正された以降も使えると思いますが、あるいは周辺の仕様が変わって使えなくなる可能性もあります。そうなったらその時にまた、何かの形でお知らせできると思います。

バグを解消する為に、ダブル・バイトのユニコード・キャラクターをシングル・バイト毎に取り扱う、少し特殊なファンクションを使っています。２つのファンクション「UnicodeLineOffset」と「getUnicodeMenuItem」は、スタック・エディターにそっくりコピペしてください。これに関しては、ライン毎の説明は省かせてもらいます。

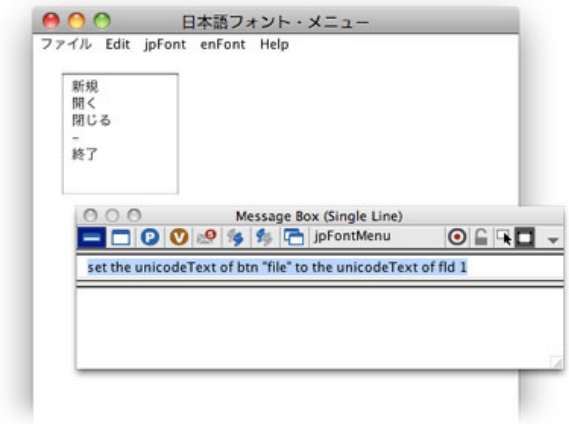
```
function unicodeLineOffset @tdata,pWhichline
   put number of characters of tdata into tlength
   set useunicode to true
   put 1 into tlinecount
   put 1 into tlineoffset
   repeat with i = 1 to tlength step 2
      if tlinecount is pWhichline then exit repeat
      if chartonum(char i to i+1 of tdata) is 10 then
         add 1 to tlinecount
         put i+2 into tlineoffset
      end if
```

```
   end repeat
   if pWhichline > tlinecount then put tlength + 1 into tlineoffset
   return tlineoffset
end unicodeLineOffset

function getUnicodeMenuItem pBtnName, pLineNum
   put the unicodetext of btn pBtnName into tdata
   put UnicodeLineOffset(tdata,pLineNum) into startchar
   put UnicodeLineOffset(tdata,pLineNum+1) - 1 into endchar
   put (char startchar to endchar of tdata) into tLineData
   put (the num of chars of tLineData + 2)/2 into tEndChar
   set the useUnicode to true
   repeat with i = 1 to (tEndChar-1)*2 step 2
      put char i to i+1 of tLineData after tString
   end repeat
   if charToNum(char -2 to -1 of tString) is 10 then
      delete char -2 to -1 of tString   --// delete return character //--
   end if
   return tString
end getUnicodeMenuItem
```

基本形は英語のメニュー・ビルダーで作るのが良いと思います。メニュー・ビルダーで新しいメニューを作ったら「New Menu」をクリックして、左側の「Edit」メニューの下に「jpMenu」と「enMenu」と言うメニューを、２つ追加してください。この２つを「日本語フォント」と「英語フォント」と言うメニューにします。「Set as stack Menu bar」のチェックを外して、MacOSのスタック上にメニューがグループ・オブジェクトになって表れるようにします。



始めにボタンのレイベルを日本語にします。LiveCodeのアイコン・メニューにある「Select Grouped」をクリックすると、グループ内のボタンがセレクトできますから、「File」をセレクトして、インスペクターのレイベルを「ファイル」にしてください。ボタンの横巾が小さいので文字が全部見えるくらいの左右にして、その他のボタンもそれに揃えて右に動かします。マウスを使ってサイズ、ポジションの修正もできますし、インスペクターの「Size & Position」からでもできます。




```
set the unicodeText of btn "file" to the unicodeText of fld 1
```

プルダウンしたときに見えるメニュー・アイテムを補助のフィールドを作って「新規、開く、閉じる、-、終了」と５行をタイプして、メッセージ・ボックスからスクリプトを送ります。４行目の「-」は英文字のハイフン（hyphen）で、区切りのラインになります。


```
set the unicodeText of btn "edit" to the unicodeText of fld 1
```

同じように「edit」ボタンもレイベルを「編集」に変えて、「カット、コピー、ペースト、削除、-、Preferences」をメニュー・アイテムにします。最後の行「Preferences」は、MacOSでは自動的にメニューのアプリケーション名の下に「環境設定」となって表れます。Windows用に「環境設定」としても良いですが、MacOSの通常の「環境設定」とは異なる「編集メニュー」の下に「環境設定」が表れます。これを避けるには、どこか（例えばボタン「edit」）にカスタム・プロパティのMac用メニュー・アイテムと、Windows用メニュー・アイテムを作って、「preOpenStack」でプラット・フォームで振り分けたメニュー・アイテムを書き込むと言う方法をとります。今は「Preferences」のまま進めます。

「日本語フォント」「英語フォント」以外は、下記のスクリプトが入ります。

```
on menuPick
   put the menuHistory of me into tLineNum  -- 選ばれたアイテムの番号
   put the short name of me into tMenu  -- メニュー・ボタンの名前
   put unicode getUnicodeMenuItem(tMenu,tLineNum) into fld 1
end menuPick
```

「the menuHistory of me」で、ユーザーが選んだメニュー・アイテムの番号を得ます。「the short name of me 」でメニューにしているボタンの名前を得て、この２つをファンクション「getUnicodeMenuItem」のパラメータにして選ばれたユニコードのメニュー・アイテムを、フィールド１に入れます。

実際のメニューでは、「getUnicodeMenuItem」が必要な事はフォント・メニュー以外ではあまりないかもしれません。下に書いたスクリプトのように「switch」を使って、メニュー・ヒストリーで得た番号を「case」で割り振る事になるでしょう。

```
switch tLineNum
      case "1"
         -- それぞれのステートメントが入ります。
         break
      case "2"
         -- それぞれのステートメントが入ります。
         break
      case "3"
         -- それぞれのステートメントが入ります。
         break
      case "5"
         -- それぞれのステートメントが入ります。
         break
end switch
```

メニュー「日本語フォント」は、ユーザーによってインストールしているフォントが違うので、メニューがマウスダウンされた時に、日本語フォントのリストをメニュー・アイテムにします。21章で日本語のフォント名だけのリストを作ったのと同じに、ファンクション「fontNames」でインストールしてあるフォント名のすべてをラインで分けたリストで返しますから、リピートで「fontLanguage」が「japanese」だけを選り分けます。

サンプルで作った日本語フォント・メニューを、メッセージ・ボックスからダウンロードしてください。

```
go to \
    url "http://kenjikojima.com/livecode/download/jpFontMenu03.livecode"
```

メニュー「日本語フォント」に入れる「mouseDown」と「menuPick」のスクリプトです。

```
local lFontNames
on mouseDown
   get the fontNames
   repeat for each line tLine in it
      if the fontLanguage of tLine is "japanese" then
         put tLine & cr after tFontNames
      end if
   end repeat
   if the platform is "MacOS" then
      sort tFontNames
   else
```

```
      put "----------------------" &cr before  tFontNames
   end if
   delete char -1 of tFontNames
   put tFontNames into lFontNames
   set the text of me to uniencode(tFontNames,japanese)
end mouseDown

on menuPick
   put the menuHistory of me into tLineNum
   put the short name of me into tMenu
   put getUnicodeMenuItem(tMenu,tLineNum) into tFontName
   put unicode tFontName & the cSelected of owner of me into fld "tempo"
   set the textFont of fld "sampleText" to line tLineNum of lFontNames
end menuPick
```

かなりの部分は21章で使ったスクリプトと同じですが、プラットフォーム（platform）の違いでWindowsではリストの始めに「"----------" &cr 」を入れています。実はユニコードにしたフォント名をメニュー・ボタンにセットすると、始めの行に不明なキャラクター４文字が付いてしまいます。「menuPick」のバグが修正された時点で、再度確認しなくてはいけません。そのため始めの行を無効にする臨時的な処置を採っています。「local lFontNames」にしているのは、メニューピック「menuPick」されたフォント名を、「menuHistory」で得た番号からラインを特定するためのものです。

日本語サブメニュー

もう少し実際のメニューに即した、サブメニューを使うテキスト・メニュー改良形を作ってみました。メッセージ・ボックスから下のスクリプトを送ってスタックを見てください。

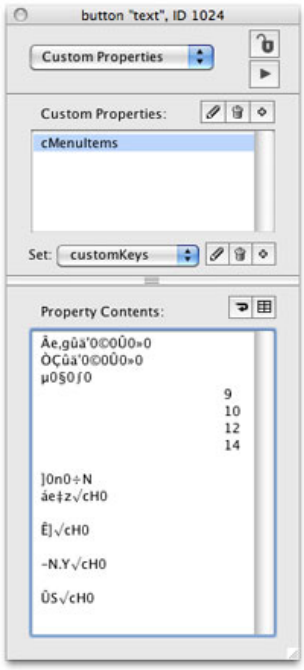
```
go to \
    url "http://kenjikojima.com/livecode/download/jpFontMenu04.livecode"
```

上で作ったメニューに「テキスト」メニューを追加しています。「テキスト」メニューをクリックすると、メニューにそれぞれサブのメニューが付いています。英語のサブメニューを思い出してください。サブ・メニューの始めにすべてタブ（tab）を入れました。

サンプル・スタックの右側に、「日本語フォント」「英語フォント」「サイズ」と行替えをし

てメニュー・アイテムの基本形を書いたフィールドがあります。「サイズ」から下の数字の部分はサブメニューで、数字の始めにタブ（tab）を入れています。その下の「文章揃え」はメニューで、やはり下にはタブを始めに入れたサブメニューを３つ置いています。「日本語フォント」「英語フォント」のサブメニューは、ユーザーによってインストールしてあるフォントが違うので、ここには書きません。この基本形のメニュー・リストを予めメニュー（ボタン名「text」）に「cMenuItems」と言う名前のカスタム・プロパティにして、メニューがマウスダウン（mouseDown）された瞬間にフォントのサブメニューを加えて、メニューに表示されるようにします。メニューの項目が入っているフィールド名を「textMenuItems」として、ユニコードのカスタム・プロパティ「the cMenuItems of btn "text"」にします。

```
set the cMenuItems of btn "text" to the unicodeText of fld "textMenuItems"
```



普通の英語メニューで使うメッセージ・ハンドラー「menuPick」のパラメータ「pWhich」は、現在のバージョンではユニコードを受け取ることができません。しかしユニコードのメニュー・アイテムの後にスラッシュ「/」を入れておく事で、スラッシュより後のキャラクターはメニュー・アイテムとして表示されませんが、英数字のキャラクターはパラメータ「pWhich」で受け取る事ができます。ボタン「text」（メニュー「テキスト」）に入れるスクリプト全文です。

```
local lMenuItems,lFontArray, lFontNum

on mouseDown
   -- メニュー・アイテムの基本形のカスタム・プロパティ
   put the cMenuItems of me into tMenuItems

   get the fontNames
   -- フォントのリストから日本語フォントだけを抜き出す
   repeat for each line tLine in it
      if the fontLanguage of tLine is "japanese" then
         put cr& tab & tLine after jpFontNames
```

```
      end if
   end repeat
   if the platform is "MacOS" then
      sort jpFontNames
   end if

   get the fontNames
    -- フォントのリストから英語フォント（ansi）だけを抜き出す
   repeat for each line tLine in it
      if the fontLanguage of tLine is "ansi" then
         put cr& tab & tLine after enFontNames
      end if
   end repeat
   if the platform is "MacOS" then
      sort enFontNames
   else
      filter enFontNames without "@*"
      sort enFontNames
      put cr before enFontNames
   end if

   -- 日本語英語フォント名のすべて
   put  jpFontNames & cr & enFontNames into allFontNames

   -- フォント名が引き出せるアレイ「lFontArray」を作る
   put 1 into tNUm
   repeat for each line tLine in allFontNames
      put tLine into lFontArray[tNum]
      add 1 to tNum
   end repeat
   put tNum into lFontNum  -- アレイ「lFontArray」で使ったの最大の数字

   -- ハンドラー・メニューピック（memuPick）でも使えるように
   -- 「lFontArray」「lFontNum」をローカルに設定しています

   -- 始めにメニューの基本形のライン２の後に英語フォント（サブメニュー）を入れる
   put uniEncode(enFontNames,Japanese) after line 2 of tMenuItems

   -- 次ににメニューの基本形のライン１の後に日本語フォント（サブメニュー）を入れる
   put uniEncode(jpFontNames,Japanese) after line 1 of tMenuItems

   -- メニューとサブメニューのすべてに「/」と数字を振ります
   repeat with i=1 to the num of lines of  tMenuItems
      put uniEncode("/") & uniEncode(i) after line i of tMenuItems
   end repeat

   -- メニューとサブメニューのすべてにをローカルに設定
   put tMenuItems into lMenuItems

   -- メニューにセットする
   set the unicodetext of me to tMenuItems
end mouseDown
```

```
on menuPick pWhich

   -- マウスダウンでタテ棒「|」の後に振った数字を「pWhich」にする
   if pWhich contains "|" then \
            delete char 1 to offset("|",pWhich) of pWhich

   put the short name of me into tMenu  --ボタン名をtMenuに入れる

   -- スタック・エディターに書いたファンクション「getUnicodeMenuItem」で
   -- 「pWhich」が何なのか、ユニコードで「tItem」に入れる
   -- フィールド「tempo」に入れるためだけなので、必ずしも必要ではない
   put getUnicodeMenuItem(tMenu,pWhich) into tItem

   if pWhich <= lFontNum then
   -- 「pWhich」がフォントに振った数字より小さければ、フォントが選ばれているので
   --  「lFontArray」からフォント名を得て「tFontName」に入れる

      put lFontArray[pWhich] into tFontName
      replace tab with "" in tFontName  -- 「tFontName」のタブを取り除く

      -- 選ばれたフォントにフィールドをセットする
      set the textFont of fld "sampleText" to tFontName

      -- フィールド「tempo」にフォント名が書き込めるようにユニコードにする
      -- フィールド「tempo」に入れるためだけなので、必ずしも必要ではない
      put uniEncode(lFontArray[pWhich],japanese) into tItem
      if tItem contains uniencode(tab) then
         replace uniencode(tab) with "" in tItem
      end if

   else
      -- フォント名以外のメニュー・アイテムが選ばれた

      -- 「pWhich」の数字からフォントの総数を引いた残りを「case」で振り分ける
      put pWhich - lFontNum into caseNum

      switch caseNum
         case 1
         case 2
         case 3
         case 4
            -- 1〜4までの「case」ならばサイズに当てはまるので
            -- アレイ「sizeArray」を作り1〜4までの「caseNum」でサイズを特定する
            put "9,10,12,14" into sizeArray
            split sizeArray by comma
            set the textSize of fld "sampleText" to sizeArray[caseNum]
            break

         case 5
            -- ask "Please Type Font Size."
```

```
            -- 日本語のダイアログ「テキスト・サイズをタイプしてください。」を出す
            ask <p>&#12486;&#12461;&#12473;&#12488;&#12539;&#12469;" & \
              "&#12452;&#12474;&#12434;&#12479;&#12452;&#12503;&#12375;" & \
                "&#12390;&#12367;&#12384;&#12373;&#12356;&#12290;</p>"

            if it <> empty then
               if isNumber(it) then
                  -- フィールドのテキスト・サイズを設定
                  set the textSize of fld "sampleText" to it
               else
                  beep
               end if
            end if
            break

         case 7
         case 8
         case 9
            -- 「caseNum」7〜9は文字揃え。
            -- アレイ「alignArray」を作って「caseNum」から6を引いた数で得る
            put "left,center,right" into alignArray
            split alignArray by comma
            set the textAlign of fld "sampleText" to alignArray[caseNum - 6]
      end switch
      if tItem contains uniencode(tab) then \
             replace uniencode(tab) with "" in tItem
   end if
   -- put tItem &cr& caseNum
   put unicode tItem into fld "tempo"
end menuPick
```

この章で新しく出て来た言葉

htmlText
プロパティ（property）　フィールドの文章をHTMLタグで表示させる
put the htmlText of fld 1

統合開発環境：LiveCodeプログラミング 初心者開発入門

無料オープンソース版「LiveCode Community Edition」の初級から中級のチュートリアル。

23: 日本語（ユニコード）の実践、その他

この章の概略

- *ユニコードは幾つかの点をきちんと押さえてスクリプトを書き進めれば、問題はいつも解決されます。その実践として、句読点で区切られたデータを日本語テーブルにして、セルを他のデータに入れ替えます。*
- *ユーザーの使っているメーラーから日本語のメールを送る書類を作成します。*
- *Windowsの日本語配列のキーボードで起こる不具合を修正します。*
- *日本語のツール・チップをコントロールに作ります。*
- *MacOS上で、日本語ファイル名の書類を保存します。*
- *MacOS上で、日本語ファイル名の書類をインポートします。*

日本語のテーブル・セルを差し替え

今までやってきた日本語（ユニコード）の扱いの応用です。
基本的な事を押さえておけば、対処できると言う例で作りました。重要な点は
１）テキスト・フィールドやボタンのレーベルの日本語表記では、ダブルバイトとシングルバイト・キャラクターを一緒にして使わない。
２）スクリプトを書いている時、今バリアブルではダブルバイトか、シングルバイトのキャラクターのどちらを扱っているのか、または文字コードは何なのかを、確実に把握しながら次の行を書いて行く。
３）バリアブル内で、ダブルバイトとシングルバイトのキャラクターを同時に扱うような場合、フィールドやボタンのレーベルに日本語を表示する前に、すべて統一したバイトのキャラクター、文字コードに戻す。

この３点に特に注意を払って書き進めて行けば、問題なく日本語の表記ができます。３）に関しては、ここで具体例を示します。

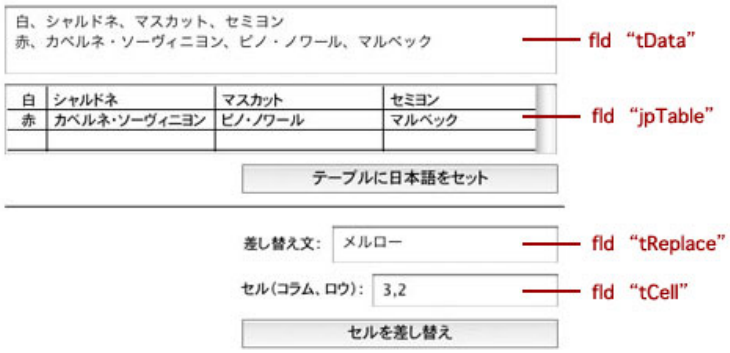
句読点と改行で区切られた日本語のデータからテーブルを作って、そのセルの日本語を他の日本語と差し替えます。簡単なワイン・リストの日本語データを、フィールド「tData」に作ってください。

白、シャルドネ、マスカット、セミヨン
赤、カベルネ・ソーヴィニヨン、ピノ・ノワール、マルベック

テーブルの横のセルを区切るのは「tab」ですから、テーブルにするには日本語句読点を「tab」にしなくてはいけません。テンポラリーのフィールド「tempo」を作って、日本語の句読点だけを打ち込んだら、句読点のUTF16のナンバーを、メッセージ・ボックスに書き出します。

```
get the unicodeText of fld "tempo"
set the useUnicode to true
put charToNum(it)
```

ユニコードのナンバーは「12289」が書き出されます。スクリプトに直接日本語の句読点を書き込めないので、日本語データの中のこのナンバーの文字を「tab」に差し替えてから、テーブル・フィールドに流し込みます。必要なフィールド、テーブル、ボタン等を図のように作ります。



一番上にフィールド「tData」を、その下にベーシック・テーブル・フィールド「jpTable」を作りました。テーブルはまだデータも何も入っていません。今はセルの横巾を設定するタブ・ストップ（tabStops）がデフォルトの「75」ですから、この図のようにはなっていりません。フィールド「tData」に、上にあるワイン・リストを書き込んでください。ボタン「テーブルに日本語をセット」のスクリプトは以下のようになります。

```
on mouseUp
   get the unicodeText of fld "tData"
   set the useUnicode to true
   replace numToChar(12289) with uniencode(tab) in it
   put unicode it into fld "jpTable"
   set the tabstops of fld "jpTable" to 26,160
end mouseUp
```

始めにフィールド「tData」から、ユニコード・テキストで日本語文字列をゲットして「it」に入れます。ナンバーからユニコードのキャラクターに変換できるように、「the useUnicode」を「true」にします。スクリプト中で扱えるように、「it」に入っているユニコード・データの句読点を「numToChar(12289)」として、ユニコードのタブ（uniencode(tab)）と入れ替えます。それから通常のユニコードをテキスト・フィールドに入れる方法で、フィールド「jpTable」に「it」を入れ、タブストップを始めのコラムを26、２番目のコラムから160に広げます。ボタン「テーブルに日本語をセット」をクリックすると、テーブルは上の図のようになります。

フィールド「tReplace」には差し替える日本語を入れます。フィールド「tCell」はセルの位置、アイテム１は「コラム（column）」、アイテム２は「ロウ（row）」の数字を入れます。この例では、現在テーブルにある２段目の左から３番目「ピノ・ノワール」を「メルロー」に差し替えます。

スクリプトの中でラインやアイテムが扱えるのは、ANSI文字だけですから、テーブルの文字列

をすべてユニコードで受け取って、ユニコードの「tab」や「cr = return」にあたるユニコード文字をANSI文字に差し替えて、ラインやアイテムを割り出します。バリアブル「tData」の文字を入れ替えたら、またタブと改行をもとのユニコードにして、テーブルに戻します。

```
on mouseUp
   put the unicodeText of fld "jpTable" into tData
   put the unicodeText of fld "tReplace" into tReplace  -- メルロー
   put item 1 of fld "tCell" into tColumn
   put item 2 of fld "tCell" into tRow
   replace uniEncode(tab) with tab in tData    -- タブをANSIに差し替え
   replace uniEncode(cr) with cr in tData      -- 改行をANSIに差し替え
   set the itemDel to tab
   put tReplace into item tColumn of line tRow of tData
   replace tab with uniEncode(tab) in tData    -- タブをユニコードに戻す
   replace cr with uniEncode(cr) in tData      -- 改行をユニコードに戻す
   put unicode tData into fld "jpTable"
end mouseUp
```

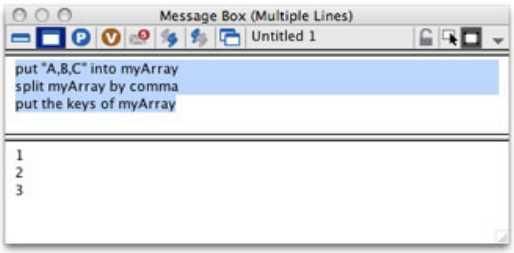
スクリプトの各ラインで扱っているキャラクターがユニコードかそうでないか、またはスクリプト中でミックスされてるキャラクターが何かを確実に把握しておけば、日本語の問題はほぼ解決されます。

日本語のアレイ

16章の英語のアレイをそのまま日本語には使えないので、日本語独自の方法が必要です。ここではスプリット「split」を使ったようなアレイを日本語で作ってみます。

まず英語のアレイを見てみましょう。

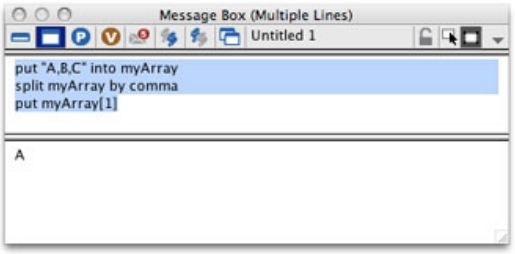
```
put "A,B,C" into myArray
split myArray by comma
put the keys of myArray
```



番号のキー（key）を使って、アレイ（array）からエレメント（element）が取り出せます。

```
put "A,B,C" into myArray
split myArray by comma
put myArray[1]
```

「A」が返されます。



ここまでがのスプリット「split」を使った英語のアレイです。日本語（ユニコード）では、スプリット「split」を使ってアレイを作れないので、同じ内容をリピートでスクエアブラケットに数字（キー key）を入れたアレイに、ユニコード文字列(エレメント element）をひとつづつ入れて行く方法を取ります。16章の『アレイ「array」の基礎』の始めに説明があります。しかしユニコードは１つのキャラクターがダブルバイト文字なので、始めの文字から２文字づつ（２ステップ）区切り文字（この場合はカンマ）かどうかチェックしながら、アレイにエレメントを入れて行きます。ちょっと面倒ですが、スクリプを理解して書いてしまえば、後はすべてLiveCodeがやってくれます。下のデータをフィールド１（fld 1）に入れます。

トマト,たまねぎ,かぶ,ばれいしょ,ピーマン,山菜,さやえんどう,グリーンピース,
そらまめ,アスパラガス,こまつな,セロリー,にら,ふき,キャベツ

データはと取り敢えずカンマ（comma）で区切りましたが、読点（、）でも他の文字でも可能です。このデータを「uncodeArray」と言うアレイにして「uncodeArray[5]」のように番号のキーでエレメントをフィールド２（fld 2）に書き出します。書き出すためのフィールド２（fld 2）を作ってください。

```
on mouseUp
   -- フィールドにある日本語データをバリアブル「tData」に入れます
   put the unicodeText of fld 1 into tData

   --  「tData」のすべての文字数を「tLength」に入れます
   put the length of tData into tLength

   put 1 into tKey  -- 始めの番号のキーを作ります

   -- charToNum がユニコードで使えるようにします
   set the useUnicode to true

   --  「tData」の文字数だけ２ステップでリピート
   repeat with i=1 to tLength step two

      if charToNum(char i to i+1 of tData) is 44 or \
            charToNum(char i to i+1 of tData) is empty then
      -- もし２文字のユニコードの番号が「44（カンマ）」
      --    またはエンプティ（最後のエレメント）ならば

         put tElement into unicodeArray[tKey]   --アレイにエレメントを入れる
         add 1 to tKey      -- キーの番号を１上げる
         put empty into tElement      -- バリアブル「tElement」を空白に
```

```
      -- そうでなかったら
      else
         -- ２文字だけバリアブル「tElemnt」の後に付ける
         put char i to i+1 of tData after tElement
      end if   -- イフ構文終わり

   end repeat   -- リピート文終わり

   -- キー「5」の「nicodeArray」をフィールド２に書き出す
   put unicode unicodeArray[5] into fld 2
end mouseUp
```

この例文では「unicodeArray[5]」を書き出すようにしていますが、サンプル・スタックはカスタム・コマンド「createUnicodeArray」でアレイを作って、「the keys」のボタン・メニューからエレメントが引き出せるようにしています。

```
go to url "http://kenjikojima.com/livecode/download/unicodeArray.livecode"
```

日本語のメールを作る

ユーザーが使っているデフォルト・メーラーで新規メールを作って、宛先、件名、本文等をレボルーションから書き込む、「revMail」という英文の為のコマンドがあります。シンタックスは

```
revMail address, [ccAddress, [mailSubject, [messageBody]]]
```

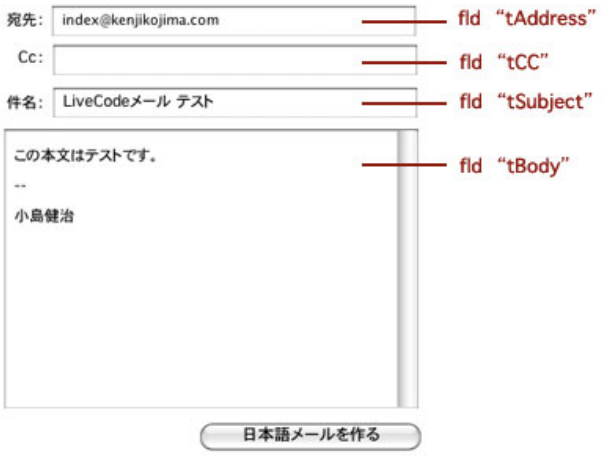
具体的には、

```
revMail index@kenjikojima.com,  , "My LiveCode Mail", fld "messageBody"
```

となります。「ccAddress」は特になかったら、空白にしてカンマ（comma）だけで、次のメール・サブジェクトをつなぎます。最後は本文の描いてあるフィールドの名前です。短い文でしたらカンマで囲ったストリングを書く事もできます。これは英文の為のコマンドなので、日本語では文字化けになります。

日本語（ユニコード）のためには「revMailUnicode」と言うコマンドがあって、シンタックスもほとんど同じです。

```
revMailUnicode address, [ccAddress, \
         [unicodeMailSubject, [unicodeMessageBody]]]
```



件名と本文はユニコードにしなくてはいけませんが、宛先、Cc は必ず英文にします。「日本語メールを作る」に入るスクリプトは以下のようになります。

```
on mouseUp
   put fld "tAddress" into tAddress
   put fld "tCc" into tCc
   put the unicodeText of fld "tSubject" into tSubject
   put the unicodeText of fld "tBody" into tBody
   revMailUnicode tAddress,tCc,tSubject,tBody
end mouseUp
```

アドレス・フィールドと日本語配列のキーボードの問題を、この後に書きます。

日本語配列のキーボード

日本語配列のキーボードを使っていると、LiveCodeのテキスト・フィールドにキーボード通りの記号等が打ち込めないキーがあります。この問題の回避は非常に簡単で、日本語のフォントをフィールドに設定すれば解決できます。特に理由がない限り、MacOSでしたら「Osaka」、Windowsでしたら「Tahoma」が無難な選択でしょう。

注意しなければいけない点は、すべてのフィールドを、あらかじめ日本語ユニコードの書体、たとえば「osaka」にしていると、ユーザーがメールアドレスなどを打ち込んで、それをスクリプト内で使いたい場合は、必ず「unidecode（ユニコードの文字列,ANSI）」または「unidecode（ユニコードの文字列,Japanese）」に変換しないと、スクリプトは正しく働きません。

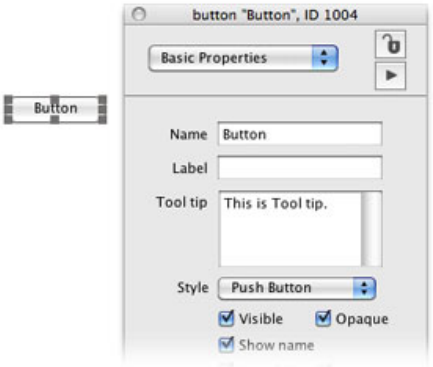
日本語以外のユニコードの言語に関しては、不明な点があるので、私の場合、多国語ユニコード対応のアプリケーションでは、文章を書き込むフィールドは、記号等の入力をしても、キーボード表示との違いが起きないユニコードの書体設定をして、Eメールアドレスを入力させるテキストボックに関しては、日本語配列キーボードのためだけの、強制変換をするようにしています。具体的には、日本語配列キーボードは、「P」の右隣をたたくと「@」が入力されなくてはいけないのですが、LiveCodeデフォルトのANSIIフィールドでは「[(左カギカッコ)」が、入力されてしまいます。これを「[(左カギカッコ)」が打ち込まれたら、直後に「@」に変えます。Eメールアドレスで使用できる記号は「@」だけですから、特にユーザーが気付くような大きなトラブルに はなりません。

Eメールアドレス用の、フィールド内に入れるスクリプトは以下の通りです。このフィールドの書体はデフォルトのままで、日本語ユニコードにはしません。

```
on returnInField
   exit to top
end returnInField

on enterInField
   exit to top
end enterInField

on keyUp pKey
   if pKey is "[" then
      replace "[" with "@" in me
      get offset("@", me)
      select after char it of me
   end if
end keyUp
```

日本語のツール・チップ

カード上にあるコントロールに、マウス・オーバーするとツールチップが現れるようにするには、プロパティ・インスペクターのツールチップ「Tool Tip」に表示させたい文を書き込みます。



しかし残念ながらプロパティ・インスペクターのツールチップに日本文を書き込むと、「?????????」になってしまいます。

スクリプトでボタン１（btn 1）に、英語のツールチップを書き込むには、

```
set the tooltip of btn 1 to "This is Tool tip."
```

日本語でツールチップを書き込むのは、英語のメール作成の時のバリエーションと非常に良く似ていて、「the tooltip」の変わりに「the unicodeTooltip」を使います。シンタックスは

```
set the unicodeTooltip of object to unicodeString
```

具体的に書いてみましょう。補助に日本語を書き込むフィールド「tempo」を作って、日本語を書き込みます。ボタン１（btn 1）に日本語のツールチップを書き込むスクリプトは、メッセージ・ボックスから送ります。

```
set the unicodeTooltip of btn 1 to the unicodeText of fld "tempo"
```

日本語ファイル名の書類を保存

LiveCodeの日本語の最大の問題は「日本語ファイル名」が扱えない事ですが、MacOSだけの開発でしたらアップル・スクリプトを応用する事で、問題は一部解決します。日本語をテキスト・フィールドから、日本語ファイル名にして保存する方法と、その日本語ファイル名の書類を、LiveCodeのフィールドにインポートする例をこれから書きます。

アップル・スクリプトを使うのは、コマンド「do」を使います。シンタックスは

```
do theAScript as applscript
```

イタリックの「theAScript」と書いた部分に、アップル・スクリプトのコードが入ります。日本語名のファイルで保存するアップル・スクリプトは下記のようになります。途中にファイルの中身の文章を書き込むので、２つのパート「cAScript1」と「cAScript2」に分けて、ボタンのカスタム・プロパティとします。

```
-- カスタム・プロパティ「cAScript1」としてボタンに保存
set newSaveFile to (choose file name with prompt
  "Save As Japanese File Name:" as Unicode text default name
     "Japanese File Name.txt" as Unicode text)
-- ここまでは改行なしの１行で書き込む
put      -- この最後に１スペースのアキを取ってください。
```

put の後にクオートで囲った、保存する文章が来ます。さらにそのあと

```
-- カスタム・プロパティ「cAScript2」としてボタンに保存
--  １行目の始めに１スペースのアキが必要です。
 into theText
```

```
set fh  to open for access newSaveFile with write permission
try
write theText to fh starting at 0
on error errMsg number ERRNO
close access fh
error "(" & ERRNO & ")" & errMsg
end try
close access fh
tell application "Finder" to update newSaveFile
```

カスタム・プロパティ「cAScript1」にある"Save As Japanese File Name:"と"Japanese File Name.txt"の部分は MacOS 10.6 までは、SJISで日本語に置き換えることができましたが、ユーザーのOSバージョンがわからないので英語で表記します。ファイル保存のダイアログは英語で表記されますが、ファイル名は日本語にすることが可能です。カードにフィールド「jpText」があるとして、その日本語を「SJIS」で保存するスクリプトを書きます。

```
on mouseUp
   put uniDecode(the unicodeText of fld "jpText", "Japanese") into tJp
   replace quote with "'" in tJp   -- ダブル・クオートをシングルに変換
   get the cAScript1 of me & quote & tJp & quote & the cAScript2 of me
   do it as applescript
end mouseUp
```

保存する日本語の中にあるダブル・クオートをそのままにしておくと、エラーになってアップルスクリプトが働きませんから、シングル・クオートに変換します。出て来るダイアログには日本語ファイル名を書き込む事ができます。

日本語ファイル名の書類をインポート

「日本語ファイル名の書類を保存」とは反対に、日本語ファイル名の書類をLiveCodeのフィールドにインポートします。上でSJISで保存したので、それがインポートできるようにしますが、UTF8でもUTF16でも同じアップル・スクリプトを使ってインポートできます。アップル・スクリプトの部分だけを下に書きます。

```
choose file with prompt ""
set openFile to result
try
read openFile
on error
close access openFile
display dialog "Error is occured."
return
end try
```

上記のアップルスクリプトをフィールド「tempo」に書き込んで、メッセージ・ボックスからボタン名「importJp」のカスタム・プロパティ「cAScript」にします。

```
set the cAScript of btn "importJp" to fld "tempo"
```

ボタン名「importJp」にカスタム・プロパティ「cAScript」を作ったら、中に以下のスクリプトを書き込みます。

```
on mouseUp
   do the cAScript of me as appleScript
   put the result into tSJISText
   put char 2 to -2 of tSJISText into tSJISText   -- クオートを取り除く
   if tSJISText contains numToChar(13) then \
         replace numToChar(13) with numToChar(10) in tSJISText
   put unicode uniencode(tSJISText,Japanese) into fld 1
end mouseUp
```

２行目の「the result」で受け取った文字列は、前後がクオートで囲まれていますから、３行目でそれを取り除きます。次の行では、もしWindowsで作られた書類だとすると改行は「crlf」が使われているので、MacOSでは改行以外にもう１行別なアキの行ができてしまいます。「crlf」の部分つまり「numToChar(13)」を、MacOSの「numToChar(10)」に変換すれば、通常のMacOSの書類の改行になります。

この章で新しく出て来た言葉

revMail
コマンド（command） ユーザーのメール・プログラムに新しい英文メールを作る
```
revMail index@kenjikojima.com, , "My LiveCode Mail", fld "messageBody"
```

revMailUnicode
コマンド（command） ユーザーのメール・プログラムに新しい日本語メールを作る
```
revMailUnicode tAddress,tCc,tUnicodeSubject,tUnicodeBody
```

keyUp
メッセージ（message） ユーザーがキーから指を上げた時に送られる
```
on keyUp pKey
      ステートメントがここに来ます。
end keyUp
```

unicodeTooltip
プロパティ（property）ユーザーがオブジェクトをマウスで指した時に出す説明文
```
set the unicodeTooltip of btn 1 to the unicodeText of fld "tempo"
```

do
コマンド（command） ステートメントを実行する
```
do tStatement as applescript
```