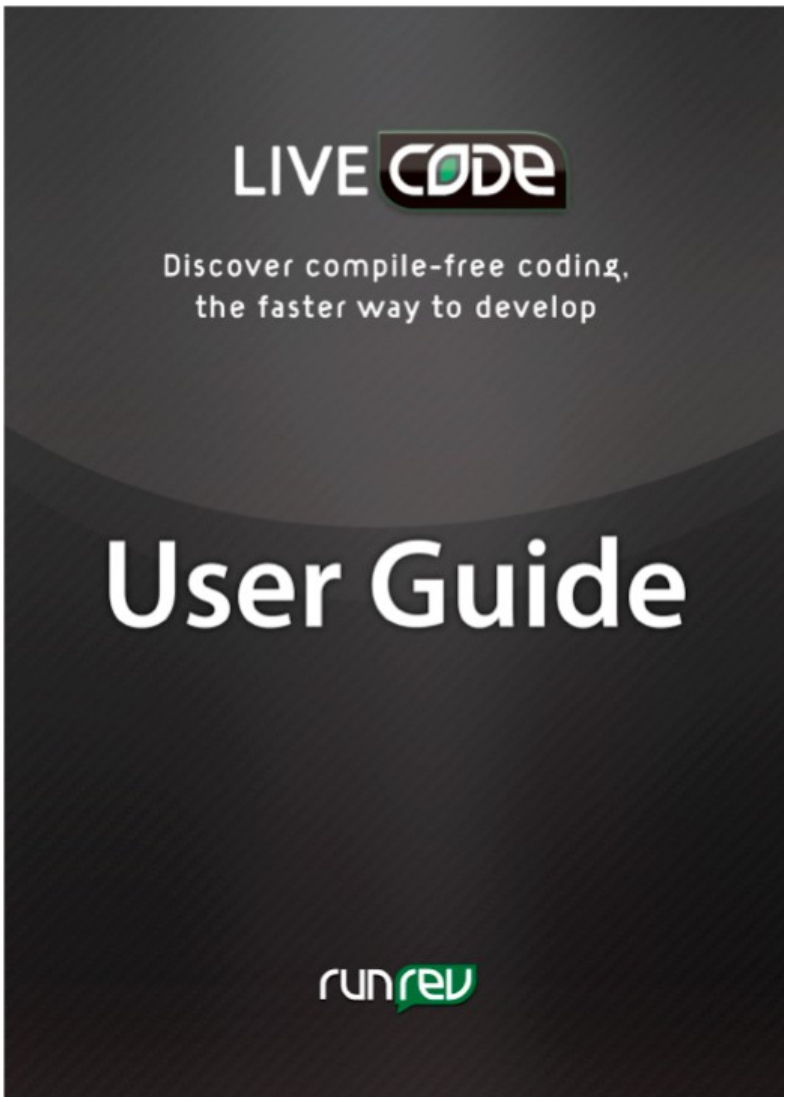


Traduction de la révision 19 (9/11/2010)

J.M. Teulé - Mars 2014

Pour faciliter la lecture et l'utilisation de cette documentation pensez à afficher la table des matières dans votre lecteur PDF

Utilisez les outils commentaire et note si vous voulez compléter ou modifier les informations
Evitez de d'imprimer autant que possible !

Chapitre 1 Introduction

1.1 Bienvenue

LiveCode facilite la réalisation de vos concepts, idées ou design d'applications performantes. Vous pouvez créer des applications de bureau ou des applications Internet ou des bases de données. LiveCode inclut un constructeur d'interface utilisateur, tout comme il permet de construire des applications sous-jacentes, sans interface utilisateur.

LiveCode est facile à apprendre. Si vous êtes un complet débutant dans le monde de la programmation, vous trouverez comment devenir rapidement productif. Si vous êtes déjà expérimenté, vous trouverez un environnement extrêmement puissant et certainement l'un des plus productif que vous n'ayez jamais utilisé.

LiveCode est un langage basé sur les objets. Ceci facilite l'écriture de modules de code rattachés directement à des objets individuels. Contrairement à d'autre langages, LiveCode vous permet d'éditer et d'exécuter une application en temps réel. Traditionnellement le cycle d'édition de compilation, de débogage, d'exécution est chronophage. Avec LiveCode tout cela peut se faire directement.

Si vous venez d'un autre langage, vous apprécierez le fait que LiveCode soit **typeless** (les données sont automatiquement stockées dans le format le plus efficient et converties) vous évitant d'avoir à apprendre comment gérer les différents types de données. La gestion de la mémoire est complètement automatique. Les applications LiveCode ne sont pas interprétées, dans le sens traditionnel, et fournissent donc d'excellentes performances. Une application graphique LiveCode offrira bien souvent un sentiment de vélocité supérieur à celui donné par l'usage d'une application Java tout en prenant moins de temps à écrire.

Une application LiveCode est totalement trans-plateforme. Ceci veut dire que vous pouvez exécuter n'importe qu'elle application créée dans tous les systèmes d'exploitation majeurs, dont Windows, Linux, Unix et Mac OS. Contrairement à n'importe quel autre **framework** trans-plateforme, une application LiveCode aura toujours un aspect d'application native de chaque système pour lequel elle sera déployée. Vous pouvez ainsi profiter des avantages propre à chacune des plateformes.

LiveCode supprime le nécessaire à connaître pour savoir comment obtenir une interface plus individualisée pour chacun des systèmes. Ceci vous épargne le temps habituellement nécessaire à apprendre des centaines d'appels d'interfaces spécifiques.

1.2 Par où commencer ?

Commencez par jeter un œil dans la section Developpers de notre site : www.runrev.com/developers

Une grande quantité de matériel s'y trouve, dont une exploration de LiveCode, des ateliers et des tutoriels, dont des leçons avec de nombreux exemples :

http://lessons.runrev.com/

Regardez également le menu **Help** de votre LiveCode. Vous y trouverez des liens vers notre site.

1.3 Recommandations Systèmes

Note : Les recommandations de mémoire et d'espace disque indiquées ci-dessous concerne unique l'IDE LiveCode et non les applications développées avec. Il est impossible de prévoir les besoins des applications que vous allez créer. C'est à vous de tester vos applications et d'indiquer à vos utilisateurs le minimum requis pour une bonne expérience utilisateur.

1.3.1 Recommandations pour les Systèmes Windows

- OS : Windows 98, ME, NT, 2000, XP, Vista, Windows 7
- Memoire : 256MB libre
- Espace Disque : minimum 100MB libre

- QuickTime (version la plus récente) est nécessaire pour les possibilités vidéo.

1.3.2 Recommandations pour les Systèmes Linux

- Linux kernel 2.4 et supérieurs
- glibc 2.2.4 et supérieurs
- X11R5 and supérieurs

  Bien qu'il soit impossible de tester toutes les distributions Linux, nous visons l'assurance que LiveCode puisse s'exécuter sur la plus grande variété de systèmes possible. Pour y arriver, le moteur a été implémenté de manière à n'avoir qu'une dépendance minimale avec le système hôte. Généralement, toute distribution Linux récente incluant Gnome/GTK possède les librairies nécessaires pour supporter toutes les fonctions de LiveCode. Par exemple Ubuntu les supporte toutes.

1.3.3 Recommandations pour les Systèmes Mac OS X

Vous pouvez développer sur :

- OS : OS X 10.2.7 or later
- Memoire: 256MB libre
- Disk space: At least 100MB libre

1.4 InstallationInstructions

Windows and Mac OS X

Double cliquez pour installer le programme.

Linux

```
cd ~
mkdir LiveCode
cd LiveCode
unzip RevEnt300Full.zip
```

À ce stade, vous devriez avoir un dossier ~ / LiveCode/3.0.0 contenant de l'IDE. Pour exécuter l'IDE simplement:

```
cd ~/LiveCode/3.0.0 ./LiveCode
```

1.5 Utiliser la Documentation

1.5.1 Documentation Conventions

Termes du langage

Cette documentation contient des exemples de syntaxe du langage LiveCode.

Le dictionnaire de syntaxe utilise les convenions standard pour signaler les différentes parties :

[ ] pour les portions optionnelles

{ } pour les alternatives à partir desquelles choisir.

| pour séparer les différente alternatives

\ caractère de continuation de ligne (cette ligne se continue à la ligne suivante).

Italiques Indique les exemples non littéraux pour la compréhension des termes.

1.5.2 Dictionnaire

Le Dictionnaire contient la syntaxe complète de LiveCode. On peut l'utiliser par le champ de recherche en haut de la fenêtre du Dictionnaire ou en utilisant la liste par types sur la gauche.

Quand le filtre de sélection est sur **ALL**, l'index des mots clés affiche tout le contenu du dictionnaire. En sélectionnant les filtres par types on limite le nombre d'entrées à celles du choix.

Au bas de chaque entrée du dictionnaire, se trouve un champ intitulé **User contributed notes**. Cet emplacement peut contenir des remarques postées par des utilisateurs à propos de la documentation proposée. C'est un avantage d'avoir un aperçu de ces notes qui offrent souvent une méthode plus simple, efficace ou pour éviter de potentiels problèmes.

Si vous voulez contribuer vous-même, ou donner un retour sur des notes déjà existantes, il faut avoir créé un compte chez **RunRev**. Une fois connecté, vous serez habilités à soumettre vos propres avis pour toute entrée du dictionnaire. Toute soumission sera mise en attente pour approbation par l'équipe **RunRev**. Vous aurez un email de retour quelque soit la décision.

Chapitre 2 Prêt à démarrer

Créer une simple application graphique avec LiveCode prends juste quelques minutes. Premièrement vous créez l'interface utilisateur, avec toutes les fenêtres, palettes, boîtes de dialogue, champs texte ou menus. Puis vous utilisez le langage de programmation de LiveCode, proche de l'anglais usuel pour déterminer le comportement de votre application.

2.1.1 Prérequis

Avant de commencer vous devez être familier de l'usage d'un ordinateur et d'applications comme des navigateurs web, traitement de texte, client email. Si vous utilisez Linux ou Unix, vous devriez être familier avec les lignes de commande sur le Terminal.

2.1.2 Programmation du contrôle des événements

Une application LiveCode est pilotée par les actions de l'utilisateur. LiveCode est constamment à l'écoute des actions communes dans l'ordinateur comme cliquer sur un bouton, saisir dans un champ, envoyer des données via le réseau ou quitter une application.

A chaque événement, LiveCode envoie un message. LiveCode envoie automatiquement chaque message à l'objet le plus approprié. Par exemple, si un utilisateur clique sur un bouton, LiveCode envoie un message au bouton. Vous placez votre code au sein du bouton de manière à lui indiquer comment répondre lorsqu'il reçoit un clic.

Il y a toute une gamme d'événements possibles. Lorsqu'un utilisateur clique sur un bouton, une série d'événements est envoyé au bouton. Par exemple, lorsque la souris pénètre la bordure du bouton un message **mouseEnter** est envoyé. Puis une série de messages **mouseMove** est généré tant que la souris se déplace sur le bouton. Lorsque le bouton de la souris est appuyé, un message **mouseDown** est émis. Lorsque ce bouton est relâché, c'est un message **mouseUp** qui apparait. Vous n'avez pas à répondre à tous ces événements. Vous devez simplement placer votre code au sein de l'objet pour contrôler les événements auxquels vous désirez que votre application réponde.

Les événements non contrôlés par les objets individuels, peuvent l'être par nombre de manières à différents niveaux de votre application, dans les librairies, ou peuvent rester ignorés. Les règles qui régissent le devenir des événements non utilisés par un objet sont décrites dans la section *Hiérarchie des Messages.*

Nous verrons plus en détail la programmation du pilotage des événements dans la section *Coder dans LiveCode.*

2.1.3 Programmation orientée objet

Toute application graphique construite avec LiveCode sera basée sur des objets. Avec LiveCode, typiquement vous créez les objets de votre application avant d'écrire tout code. Vous pouvez démarrer en dessinant les boutons, les champs textes et tout autre contrôle nécessaire à votre application. LiveCode fonctionne comme tout autre application de mise en page, de dessin ou de développement. Vous pouvez sélectionner vos contrôles par glisser/déposer, les retailler et changer leur niveau de profondeur pour les situer plus ou moins proches du niveau supérieur de votre interface.

Lorsque vos objets sont en place, vous pouvez procéder à l'écriture du code de chacun d'eux, propre à répondre aux événements désirés. LiveCode inclut un environnement graphique de développement complet qui rend aisé la création et l'édition de toute sorte d'interface utilisateur. Il inclut les objets pour tous les éléments basiques des systèmes d'exploitation (boutons, cases à cocher, champs de texte, menus, affichage et plus encore). Pour finir vous pouvez inventer et modifier vos propres objets dont l'aspect et le comportement sera comme vous le désirez.

Si vous écrivez une application non graphique, vous pouvez choisir de créer quand même les objets pour vous aider à organiser votre code au sein de sections et de rendre ensuite ces objets invisibles ou bien vous pouvez simplement écrire votre code dans un fichier texte et l'exécuter directement.

La programmation objet est traitée avec plus de détail dans la section *Construire Une Interface Utilisateur*.

2.1.4 Le mode « Edition » et « Exécution »

A la différence de la plupart des outils de développement, une application LiveCode peut être créée, éditée, déboguée et exécutée en temps réel.

En mode exécution, les objets reçoivent tous les messages normaux qui pilotent l'application LiveCode. Par exemple, cliquer sur un bouton en mode exécution entrainera l'envoi d'un message **mouseUp** à lui-même déclenchant l'exécution du script du bouton s'il est prévu que celui-ci réagisse au message **mouseUp**.

En mode édit, les objets ne reçoivent pas de message lorsque vous cliquez dessus, et vous pouvez donc déplacer, retailler ou éditer les propriétés des objets.

Il y a quelques autres différences entre ces deux modes. Vous pouvez afficher et éditer les propriétés et le code dans l'un ou l'autre des modes. Votre application ne s'arrête pas de fonctionner pendant que vous effectuez vos changements. Seule l'interaction de la souris avec le objets est interrompue dans le mode édit, facilitant vos modifications.

Grâce à ce fonctionnement en temps réel, vous pouvez aisément faire de simples changements et vérifier aussitôt leurs effets. Ceci vous permet de modeler et d'expérimenter via un processus itératif propice à une plus grande productivité et une meilleure expérience de développement.

Astuce : Pour désactiver temporairement tous les messages envoyés vers votre application pendant son édition, choisissez **Suppress Messages** depuis le menu **Development** ou la barre d'outil.

2.2 Structurer son Application

2.2.1 Piles et Fichiers

La première étape de création d'une application dans LiveCode est la création d'une fenêtre, qui dans LiveCode est appelée une pile. Chaque fenêtre visible dans LiveCode est une pile. Les palettes, boîtes de dialogue et les fenêtres standard, tout est pile.

Chaque pile contient un ou plusieurs ensembles d'informations appelés cartes. Chaque carte peut avoir différentes apparences ou bien toutes les cartes d'une pile rester d'aspect identique. En allant de carte en carte dans une pile, vous changez l'apparence de la fenêtre de la pile. Vous pouvez voir une pile LiveCode comme un jeu de cartes (d'où le nom) dans lequel vous pourriez voir toutes les cartes, mais une seule à la fois qui puisse être visible. Une pile peut n'avoir qu'une seule carte comme beaucoup.

Tous les objets (contrôleurs) de l'interface utilisateur sont créés par un glisser/déposer dans une aire de la carte.

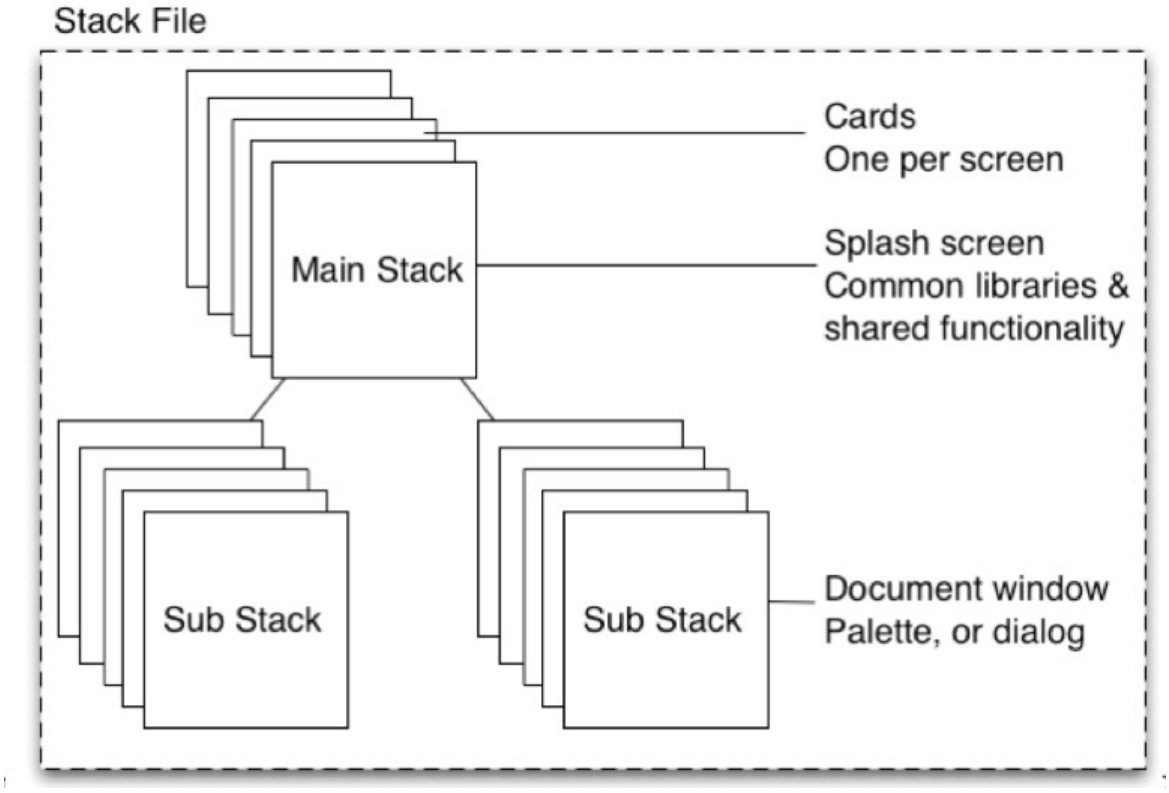
Vous pouvez aussi grouper des contrôles si vous voulez qu'ils opèrent ensemble. Par exemple, si vous avez un ensemble de boutons de navigation permettant d'aller de carte en carte dans votre pile, vous pouvez les assembler en un seul groupe. Les groupes pouvant apparaître dans plus d'une carte, vos boutons de navigation apparaitront ainsi dans chaque carte de votre pile. Pour plus de détails voir la section *Groupes et Arrière Plans*.

Une collection de piles peut être sauvegardée comme un ensemble dans un simple fichier. Ce fichier est identifié comme un fichier pile. La première pile du fichier est appelé Pile principale, **main stack**, et sera automatiquement chargée au lancement de votre application.

2.2.2 La Structure d'une Fichier Pile

Chaque fichier LiveCode contient une ou plusieurs piles : soit une simple pile principale soit une pile principale et une ou plusieurs sous-piles. Puisque chaque pile est une fenêtre (incluant les fenêtre éditables, les boîtes de dialogue modales ou non et les palettes), un simple fichier pile peut contenir de multiples fenêtres.

Vous pouvez utiliser cette possibilité pour assembler plusieurs piles reliées en un seul fichier pour faciliter la distribution, organiser vos piles en catégories ou permettre à plusieurs piles d'hériter de propriétés de la même pile principale.



Structure du Fichier de Pile

2.2.3 Ouvrir un Fichier Pile

Lorsque vous ouvrez un fichier pile, soit en utilisant **Open Stack** dans le menu **File** soit en utilisant l'une des commandes de navigation (voir **open, go, modal, modeless, palette**, ou **topLevel** dans le dictionnaire LiveCode), le fichier pile de la pile principale ouvre automatiquement sa propre première carte.

Les sous-piles dans le fichier pile ne s'ouvrent pas automatiquement lors de l'ouverture du fichier pile. Vous devez ouvrir une sous-pile depuis une commande, depuis **Message Box** ou en utilisant le navigateur

d'application, **Application Browser**.

Important: Un fichier pile est sauvé comme un tout. Si vous sauvez une pile, toutes les autres piles dans le même fichier pile sont sauvées en même temps.

2.2.4 Pile principale et Sous-piles

La première pile créée dans un fichier pile est nommée pile principale, **mainstack**. Toute autre pile créée dans le même fichier pile est appelée sous-pile, **substack**, de la pile principale.

La pile principale est une partie de la hiérarchie objet de toutes les autres piles dans le même fichier pile. En d'autres termes (en raison des héritages de propriété et des comportements partagés) la pile principale contient ses sous-piles. Les événements non pilotés par une sous-pile sont passés au script de la pile principale, la couleur et les propriétés des polices sont héritées de la pile principale par ses sous-piles. Pour plus de détails sur la hiérarchie des objets et l'héritage dans LiveCode en général, voir la section **The Message Path**.

Les boîtes de dialogue et les palettes sont communément stockées comme sous-piles de la fenêtre de l'application principale, qui est typiquement la pile principale. Ceci vous permet de stocker le code et les fonctions communes utilisés par toutes les sous-piles dans le script de la pile principale. Parce que la pile principale est une part de la hiérarchie objet de ses sous-piles, les sous-piles peuvent appeler cette fonctionnalité depuis des scripts au sein des sous-piles.

2.2.5 Piles, Fichier de Pile et Piles de mémoire

Un fichier de pile peut être chargé en mémoire sans être déjà ouvert. Une pile dont la fenêtre est fermée (pas seulement masquée) n'est listée dans la fonction **openStacks.** Toutefois elle monte en mémoire et ses objets sont accessibles pour les autres piles. Par exemple, si une pile fermée, chargée en mémoire, contient une image, vous pouvez utiliser cette image comme icône de bouton dans une autre pile.

Si une pile dans un fichier de pile est chargée en mémoire, alors c'est vrai pour toute autre pile dans le même fichier de pile. Vous ne pouvez charger une pile dans un fichier de pile sans charger le reste en même temps, même si vous n'ouvrez qu'un seule des piles.

Une pile peut-être chargée en mémoire sans être préalablement ouverte sous les conditions suivantes :

- Une partie du code d'une autre pile lit ou paramètre des propriétés au sein d'une pile fermée. Ceci charge en automatiquement en mémoire la pile référencée.
- La pile est dans le même fichier de pile qu'une autre pile ouverte.
- La pile était ouverte puis a été fermée et sa propriété **destroyStack** n'a pas été activée. Sans **destroyStack** actif, la pile est fermée mais non déchargée lors de la fermeture de la fenêtre.

2.2.6 Media & Ressources

En élaborant un projet, il est important de considérer à quels types de media vous devrez accéder et comment structurer son accès.

LiveCode supporte une large étendue de formats de media. Ces médias peuvent être accessibles en utilisant le support intégré, à travers QuickTime ou via une librairie externe. L'avantage d'utiliser le support intégré est que vous pouvez toujours afficher ou lire les médias sur toutes les plateformes sans avoir à vérifier que tout composant de tierce partie a été installé. L'avantage d'utiliser QuickTime est qu'un plus large éventail de médias est pris en charge. D'autres bibliothèques tierces peuvent permettre une plus grande gamme d'accès aux médias.

Tip: Pour manipuler une fenêtre dans une librairie externe, utilisez la proporiété **windowID**. Pour plus d'information, voir le Dictionnaire de LiveCode.

Comme chaque fichier de pile chargé prend autant de mémoire que la taille de l'ensemble de ses piles, il est

souvent conseillé de placer de grands objets rarement utilisés (tels que des photos de couleur, son et vidéo) dans des fichiers externes, qui peuvent être livrés à votre demande et chargés en mémoire seulement lorsque vous en avez besoin.

La prise en charge intégrée des médias vous permet d'incorporer des médias directement au sein de votre fichier de pile, ou via une référence externe, le stockant dans un dossier de données, en ligne ou sur CD. QuickTime doit être situé à l'extérieur et peut être soit local ou en streaming depuis un serveur.

Intégration des médias au sein de votre projet	*Médias par référencement externe*
Permet la distribution d'une application mono-fichier pour faciliter une distribution sûre.	0
Nécessite l'importation de médias chaque fois qu'il est mis à jour.	0
Nécessite assez de mémoire pour charger tous les médias.	0
Vous permet d'utiliser directement les fonctions intégrées d'édition.	0
Est moins pratique pour la création de grandes applications thématiques ou localisée où un ensemble de supports est remplacé par un autre.	0

Astuce : Lorsque vous importez des images, utilisez la bibliothèque d'images, ou créez une carte «bibliothèque» qui contient tous les originaux, puis référencez ces objets tout au long de votre projet. Pour plus de détails sur les images de référencement, voir la section sur l'objet de bouton.

Pour plus de détails sur les formats d'image supportés nativement, voir la section sur l'objet **Image**. Pour plus de détails sur les formats audio supportés nativement, voir la section sur l'objet **audioClip**. Pour plus de détails sur la façon de contrôler un film QuickTime, consultez la section sur l'objet **Player**.

2.2.7 Comment Utiliser des Fichiers Externes

Il y a trois principales façons d'utiliser des fichiers externes :

Stocker les médias tels que des images, des sons et vidéo dans des fichiers externes, dans le format approprié, et utiliser les commandes référencées pour afficher le contenu des fichiers. Lorsque vous utilisez une commande référencée, le fichier image, le son ou la vidéo est chargé en mémoire que si une carte qui contient le contrôle référencé est affichée. La mémoire nécessaire pour l'image, le son ou la vidéo est donc uniquement utilisée lorsque réellement nécessaire. En utilisant les chemins de fichiers relatifs, vous pouvez utiliser l'application et ses fichiers de données dans n'importe quel autre système. Pour plus de détails sur l'utilisation des chemins de fichiers, voir la section sur les Spécifications des noms de fichiers et chemin d'accès..

Remarque : Pour créer une commande référencée, utilisez le sous-menu "**NewReferencedControl**" dans le menu **File**, ou créez une image vide ou un objet **player**, puis définissez la propriété **fileName** de l'objet à un chemin de fichier pour le fichier que vous souhaitez référencer. Pour plus de détails, voir la section sur la construction d'une interface utilisateur.

Garder des parties de votre application dans un fichier de pile séparée, et se référer aux piles dans ce fichier de pile, au besoin. La propriété **stackFiles** simplifie référence à des fichiers de pile externes. Lorsque vous définissez la propriété **stackFiles** d'une pile pour inclure un ou plusieurs chemins d'accès, les piles indiquées à ces endroits deviennent disponibles pour votre pile simplement en se référant aux piles par leur nom.

2.2.8 Quand utiliser une Base de données

Vous n'avez pas besoin d'utiliser une base de données externe pour stocker des informations pour votre application. Vous pouvez stocker des informations dans des piles, dans des fichiers texte, et une variété d'autres fichiers, et les lire dans votre application en fonction des besoins, les modifier si nécessaire. Cependant, en règle générale, nous recommandons que lorsque vous avez plus de deux mille enregistrements d'informations, ou si vous voulez permettre à plusieurs utilisateurs d'accéder aux informations en même temps, vous envisagiez d'utiliser une base de données.

Dans ces circonstances, une base de données externe offre de nombreux avantages. Une base de données située sur votre machine locale sera rapide et efficace pour l'accès et la recherche dans les enregistrements. Une base de données située sur un serveur peut être accédée par plus d'un utilisateur. Selon sa configuration, une base de données peut être adaptée à l'accès constant par des centaines d'utilisateurs, chacun recevant et actualisant ses données continuellement. Les bases de données SQL intègrent des capacités de verrouillage des enregistrements, ce qui empêche que les modifications d'un utilisateur anéantissent celles d'un autre, une nécessité pour des bases de données multi-utilisateurs fiables. D'autres fonctionnalités intégrées travaillent en arrière plan pour s'assurer que les données dans la base ne sont pas corrompues par des interférences entre les différents utilisateurs.
Les bases de données SQL sont également construites pour la vitesse. Lors de la recherche sur des dizaines, des centaines de méga-octets, ou plus encore, les performances d'une base de données optimisée sera généralement bien meilleure que celle d'une pile pour faire la même recherche. En outre, les piles doivent être chargées en mémoire pour être détectables, et donc l'ensemble de la collecte des données doit tenir dans la mémoire locale.

Enfin, si vous utilisez une base de données externe, vous pouvez mettre le traitement lourd sur un serveur conçu à cet effet, tout en utilisant la flexibilité de LiveCode pour passer les options de l'utilisateur lors de la sélection des données, et les présenter sous une forme exploitable.

Avec la bibliothèque de base de données incluse dans LiveCode, votre application peut communiquer avec les bases de données SQL externes. Vous pouvez obtenir des données à partir de bases mono-utilisateur ou multi-utilisateur, les mettre à jour, obtenir des informations sur la structure de la base et afficher les données obtenues dans votre pile.

Pour plus de détails sur l'utilisation de bases de données, voir le chapitre 8, Travailler avec bases de données.

2.2.9 Limites mémoire de LiveCode

Le tableau suivant présente les limites de mémoire pour les différents types de composants de LiveCode. S'il vous plaît notez que ces limites se réfèrent à des maximums qui peuvent être utilisés à n'importe quel moment. Vous pouvez stocker des informations supplémentaires sur le disque ou dans une base de données et les charger quand cela est nécessaire.

Une note sur les entrées désignées comme "illimité"

Espace adressable total	4 Go (sur les systèmes 32 bits)
Longueur maximale d'une ligne dans un champ	65.536 caractères stockage
La taille maximale d'un objet	Pas plus de 32 786 pixels de large pour l'affichage
Le nombre maximum d'objets dans un groupe	illimité
Le nombre maximum d'objets dans une carte	illimité
Le nombre maximum de cartes dans une pile	illimité
Le nombre maximum d'objets dans une pile	illimité
Longueur maximale des noms d'objets	65.536 caractères
Longueur maximale des noms de propriétés personnalisées	255 caractères

Longueur maximale de la commande ou de la fonction des noms	65.536 caractères
La taille maximale des propriétés personnalisées	illimité
Le nombre maximum de propriétés personnalisées	illimité
La taille maximale d'un script	illimité
La taille maximale des autres propriétés	64K
Niveau d'imbrication maximum dans les structures de contrôle	illimité
Niveau maximum de récursivité	illimité

Une remarque sur les entrées désignées comme "illimité" : Puisque chaque fichier de pile ouvert réside entièrement en mémoire, les piles de LiveCode (et toutes les structures au sein d'une pile) sont effectivement limitées par la mémoire disponible et par le total d'espace d'adressage de LiveCode qui sera de 4 gigaoctets (4294967296 octets) sur les systèmes 32 bits, ou 16 pétaoctets (18.446.744.073.709.551.616 octets) sur les systèmes 64 bits.

Chapitre 3 L'environnement de Développement

Cette section détaille les principaux composants dans l'environnement de développement intégré de LiveCode (IDE). L'environnement de développement contient toutes les fonctionnalités dont vous avez besoin pour créer rapidement une application professionnelle. Le navigateur de l'application vous permet de trouver votre chemin autour de votre application que vous développez. L'inspecteur de propriétés vous permet de définir l'apparence et le comportement de base. L'éditeur de code vous permet d'ajouter du code pour chaque objet dans votre application. En plus de ces outils standards, **Message Box** propose une mini ligne de commande qui vous permet de développer des aspects de votre application automatiquement, ou de tester le code et les fonctionnalités de votre application.

3.1 La Barre de Menu

3.1.1 Le Menu *File*

Le ***File Menu*** contient les commandes pour ouvrir, fermer et enregistrer des fichiers, imprimer, et incorporer des fichiers dans votre pile.

New Mainstack Crée une nouvelle fenêtre **Untitled 1** dans la pile principale. Lorsque vous enregistrez la pile, LiveCode demande un nom de fichier et un emplacement.

New Substack of (main stack name) Crée une nouvelle pile sans titre dans le même fichier que la pile principale active. Lorsque vous enregistrez la sous-pile, elle est enregistrée dans le dossier de la pile principale. Cet item est désactivé si la fenêtre active n'est pas une pile principale.

Open Stack... Ouvre la pile principale correspondant au fichier que vous sélectionnez. Si vous sélectionnez un fichier HyperCard, il est automatiquement converti en une pile principale de LiveCode.

Open Recent Stack Ouvre une liste contenant les noms des 30 principales piles que vous avez le plus récemment fermées. Choisir un nom pour ouvrir la pile. Vous pouvez modifier le nombre d'éléments affichés en utilisant **Preferences**.

Close Ferme la fenêtre active. Cet item est désactivé si aucune fenêtre n'est ouverte.

Close and Remove from Memory... Ferme la pile en cours, et toutes les piles dans le même fichier de pile que la pile en cours, et supprime toutes les piles dans le fichier de la mémoire. Cet item est désactivé si la fenêtre active n'est pas une pile utilisateur.

Import as Control Ouvre une liste que vous pouvez utiliser pour choisir un fichier et placer son contenu dans un nouveau contrôleur de type approprié. Cet item est désactivé si la fenêtre active n'est pas une pile.

Image File... Importe le fichier image que vous choisissez comme une nouvelle image dans la carte actuelle. Vous pouvez importer des fichiers GIF, JPEG, PNG, BMP, XWD, XBM, XPM, ou PBM, PGM, PBM (ou les fichiers PICT sur les systèmes Mac OS X et OS). Pour plus d'informations, voir la section sur les objets image.

Snapshot... Affiche un réticule pour vous permettre de sélectionner une zone de l'écran, et importate une capture d'écran de cette région comme une nouvelle image dans la carte actuelle.

Audio File... Importe le fichier son que vous choisissez comme un nouveau clip audio dans la pile actuelle. Vous pouvez importer des fichiers fichiers WAV, AIFF, ou les fichiers AU. Pour plus de détails, voir la section sur l'objet **AudioClip**.

Video File... Importations le fichier vidéo que vous choisissez en tant que nouveau clip vidéo dans la pile actuelle. Vous pouvez importer des fichiers QuickTime, AVI, ou MPEG. Pour plus de détails, voir la section sur l'objet **videoClip**.

Text File... Importe le fichier texte que vous choisissez comme un nouveau champ de la carte actuelle.

EPS File... Importe le fichier **PostScript** encapsulé que vous choisissez comme un nouvel objet **EPS** dans la carte actuelle. Cet article est disponible uniquement sur les plates-formes Unix avec **Display Postscript** installé. Pour plus d'informations, consultez la section sur les objets **EPS**.

All Images in Folder... Importe tous les fichiers d'image dans le dossier de votre choix, et les place dans de nouvelles images sur la carte actuelle. Les sous-dossiers, et autres types de fichiers, sont ignorés. Pour plus de détails, voir la section sur l'objet image.

All Audio Files in Folder... Importe tous les fichiers audio dans le dossier de votre choix, et les place dans de nouveaux audio clips dans la pile actuelle. Les sous-dossiers, et autres types de fichiers, sont ignorés. Pour plus de détails, voir la section sur l'objet audioClip.

New Referenced Control Ouvre une liste que vous pouvez utiliser pour sélectionner un fichier à référencer (c'est à dire créer un lien) à un nouveau contrôleur de type approprié. Cet item est désactivé si la fenêtre active n'est pas une pile.

Image File... Crée une nouvelle image sur la carte actuelle et des liens vers le fichier d'image que vous sélectionnez dans le nouvel objet image. Pour plus de détails, voir la section sur l'objet image.

Quicktime-Supported File... Crée un nouveau lecteur sur la carte actuelle et relie le fichier audio ou vidéo que vous sélectionnez avec le nouveau lecteur. Pour plus de détails, voir la section sur l'objet **Player**.

All Images in Folder... Pour chaque fichier d'image dans le dossier sélectionné, crée une nouvelle image sur la carte actuelle et des liens avec chacun des fichiers. Les sous-dossiers, et autres types de fichiers, sont ignorés. Pour plus de détails, voir la section sur l'objet image.

Save... Enregistre les modifications apportées pour la pile actuelle et pour toutes les autres piles qui se trouvent dans le même fichier de pile. Si le fichier n'a pas encore été enregistré, vous devez indiquer le nom et l'emplacement du nouveau fichier. Cet item est désactivé si la fenêtre active n'est pas une pile.

Save As... Enregistre la pile actuelle, ainsi que tous les autres piles qui se trouvent dans le même fichier, un nouveau fichier avec un nom et un emplacement que vous spécifiez. Le nouveau fichier devient la copie de travail actuel.

Move Substack to File... Enregistre la sous-pile active comme une pile principale dans un fichier à part, avec un nom et un emplacement que vous spécifiez. La sous-pile est retirée de son fichier de pile précédente. Cet item est désactivé si la fenêtre active n'est pas un sous-pile.

Revert to Saved... Supprime toute modification apportée à la pile actuelle, ainsi qu'à toutes les autres piles qui se trouvent dans le même fichier de pile. La totalité du fichier pile est ensuite rechargé.

Standalone Application Settings... Réglages pour exporter la pile actuelle de la distribution comme une application autonome. Cet item est désactivé si la fenêtre active n'est pas une pile. Pour plus de détails, voir la section sur Déploiement de votre application.

Save As Standalone Application... Crée un paquet de la pile actuelle dans une application autonome à fin de distribution en utilisant les paramètres définis dans la fenêtre Paramètres de l'application autonome.

Page Setup... Ouvre la boîte de dialogue Configuration de la page de l'imprimante sélectionnée.

Print Card... Imprime la carte actuelle.

Print Field... Imprime le champ sélectionné en utilisant la commande **revPrintField**. Pour plus de détails, voir la section sur l'impression.

Exit... Ferme toutes les piles ouvertes et quitte LiveCode.

3.1.2 Le Menu Edit (Editer)

Le menu **Edition** contient les commandes pour sélectionner, couper, copier et coller du texte et des objets.

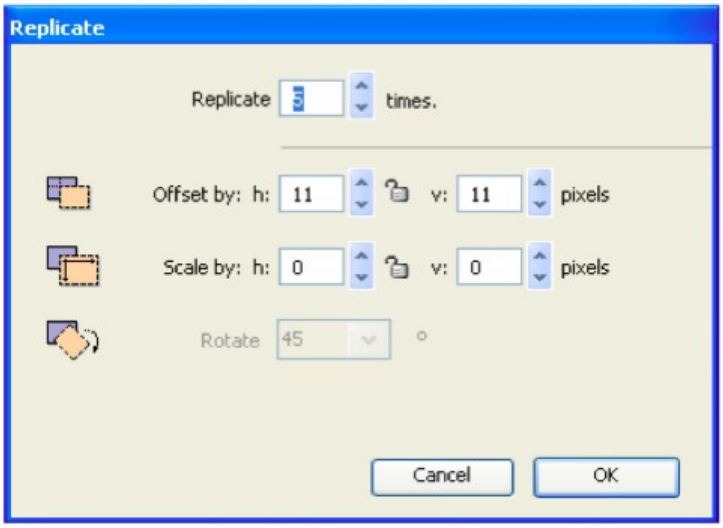
Undo Inverse la modification la plus récente de texte, de dessin, de mouvement ou de suppression d'un objet.

Cut, Copy, Paste Coupe, copie ou colle le texte ou l'objet sélectionné. Le texte et les images peuvent être échangés avec d'autres programmes de cette manière. Les objets ne sont disponibles que dans l'instance en cours d'exécution de LiveCode.

Clear Supprime le texte ou des objets sélectionnés, sans les placer dans le presse-papiers.

Duplicate Fait une copie de ou des objets sélectionnés. Si l'objet est une carte, la copie est ajoutée après la carte actuelle. Si l'objet est une commande, la copie est placée sur la carte en cours, en dessous et à droite de l'objet original. Cet item est désactivé si aucun objet n'est sélectionné.

Replicate... Donne une ou plusieurs copies du ou des objets sélectionnés en utilisant les paramètres que vous sélectionnez.



La boîte de dialogue ***Replicate*** (Répliquer) vous permet la sélection du nombre de copies de l'objet sélectionné que vous souhaitez. Vous pouvez spécifier que chaque copie est décalée d'un certain nombre de pixels à partir de la copie précédente. Vous pouvez également spécifier que chaque objet est agrandi ou rapetissé en nombre de pixels. Si l'objet est une image ou un graphique, chaque copie peut être pivotée d'un nombre déterminé de degrés.

Fenêtre Replicate

Select All Sélectionne tout le texte dans le champ actuel ou tous les contrôles sur la carte actuelle.

Deselect All Désélectionne tous les objets sélectionnés, ou supprime le point d'insertion à partir d'un champ.

Invert Selection Sélectionne tous les objets non sélectionnés et désélectionne tous ceux qui sont sélectionnés. Cet item ne fonctionne pas sur les sélections de texte.

Select Grouped Controls Si cette option est cochée, cliquez sur un contrôle qui fait partie d'un groupe ne sélectionne que ce contrôle. Si rien n'est fait, en cliquant sur un contrôle qui fait partie d'un groupe sélectionne le groupe.

Intersected Selections Si cette option est cochée, l'outil Pointeur sélectionne chaque objet qui croise le rectangle de sélection. Si elle ne l'est pas, faire glisser l'outil Pointeur sélectionne uniquement les objets qui sont entièrement entourées par le rectangle de sélection.

Find and Replace... Recherche de texte dans les champs, les propriétés, les scripts, les variables globales ou les contenus de boutons pour éventuellement le remplacer. Pour plus de détails, voir la section sur Rechercher et remplacer.

Preferences Définit les préférences de toute l'application. (Notez que cet article est disponible dans le menu

LiveCode lorsqu'il fonctionne sur Mac OS X.)

3.1.3 Le Menu Tools (Outils)

Le menu Outils contient les commandes pour travailler avec les palettes d'outils de LiveCode et pour utiliser des outils de développement de pile.

Browse Tool Sélectionne l'outil ***Browse*** pour l'exécution d'un programme. Cela vous permet d'effectuer des actions d'utilisateur telles que l'exécution de scripts en cliquant sur les boutons ou la saisie de texte dans un champ.

Pointer Tool Sélectionne l'outil Pointeur pour l'édition d'un programme. Vous permet de sélectionner, de déplacer et redimensionner des objets.

Tools Palette Affiche ou masque la palette d'outils pour choisir les outils pour la création de l'objet.

Paint and Draw Tools Affiche les outils de dessin pour dessiner des formes particulières.

Application Browser Ouvre la fenêtre du navigateur d'application, qui répertorie toutes les piles ouvertes, les cartes dans chaque pile, et les contrôles sur chaque carte. Pour plus de détails, voir la section sur le navigateur de l'application.

Project Browser Pour gérer l'ensemble d'un projet. Fonctions voisines d'**Application Browser** dans une interface modernisée.

Message Box Affiche ou masque la boîte de message. La boîte de message est un outil de ligne de commande qui vous permet d'exécuter des scripts ou d'effectuer des opérations d'édition automatisées. Pour plus de détails, voir la section sur la boîte de message **Message Box**.

Menu Builder Crée ou modifie la barre de menu de la pile active. Pour plus de détails, voir la section sur l'utilisation des menus.

3.1.4 Le menu Objet

Le menu Objet contient des commandes permettant de modifier les propriétés de l'objet ou des objets sélectionnés, de créer de nouveaux objets, et de travailler avec des groupes.

Object Inspector Ouverture de l'inspecteur pour l'objet sélectionné, vous permettant d'afficher et de définir les propriétés des objets. Si plus d'un objet est sélectionné, les modifications apportées aux propriétés sont appliquées à chacun des objets sélectionnés. Cet item est désactivé si aucun objet n'est sélectionné. Pour plus de détails, voir la section sur l'inspecteur des propriétés.

Card Inspector Ouverture de l'Inspecteur pour la carte actuelle, vous permettant de visualiser et de définir les propriétés de la carte.

Stack Inspector Ouverture de l'inspecteur pour la pile actuelle, vous permettant de visualiser et de définir les propriétés de la pile.

Object Script Ouverture de l'éditeur de code pour les objets sélectionnés. Si plus d'un objet est sélectionné, une fenêtre de l'éditeur de script s'ouvre pour chacun. Pour plus de détails, voir la section sur l'éditeur de code.

Card Script Ouverture de l'éditeur de code pour la carte actuelle.

Stack Script Ouvre l'éditeur de code de la pile actuelle.

Group Selected Intègre les objets sélectionnés dans un groupe. Cet item change pour ***Ungroup Selected*** si le seul objet sélectionné est un groupe. Pour plus de détails, voir la section sur groupes et les arrière-plans.

Ungroup Selected Eclate le groupe sélectionné en objets individuels. Cela supprime le groupe de façon permanente si vous visitez une autre carte avant de regrouper à nouveau les objets. Cet élément devient **Group Selected** si plus d'un objet est sélectionné.

Edit Group Entre en mode d'édition de groupe pour apporter des modifications aux objets dans le groupe sélectionné. Change cet item en ***Stop Editing Group*** dans le mode édition de groupe. Cet item est désactivé si aucun objet n'est sélectionné, si plus d'un objet est sélectionné, ou si l'objet sélectionné n'est pas un groupe.

Stop Editing Group Quitte le mode d'édition de groupe. Change cet item en ***Edit Group*** si la pile n'est pas déjà en mode d'édition de groupe.

Remove Group Supprime le groupe sélectionné de la carte actuelle, sans le supprimer de la pile.

Place Group Ouverture d'une liste contenant les noms des groupes qui sont dans la pile, mais pas sur la carte actuelle. Choisissez un groupe pour l'ajouter à la carte actuelle. Cet item est désactivé si tous les groupes de la pile figurent déjà sur la carte actuelle, ou s'il n'y a pas des groupes. Note : Seuls les noms des groupes de haut niveau sont répertoriés ; les groupes qui font partie d'un autre groupe ne sont pas répertoriés dans ce menu.

New Card Crée une nouvelle carte en suivant la carte actuelle. Note : S'il y a des groupes partagés sur la carte actuelle lorsque vous choisissez cette option, ils sont placés automatiquement sur la nouvelle carte. Si le **backgroundBehavior** des groupes est désactivé, ils ne sont pas placés automatiquement sur les nouvelles cartes.

Delete Card Supprime la carte actuelle de la de pile la plus élevée.

New Control Ouverture d'une liste que vous pouvez utiliser pour créer un nouveau contrôleur. Pour plus de détails, voir la section sur les types de contrôle.

Flip Ouverture d'une liste que vous pouvez utiliser pour changer l'orientation de l'image sélectionnée ou du graphique. Cet item est désactivé si un objet autre qu'une image ou un graphique est sélectionné.

> **Horizontal** Permute les bords gauche et droit de l'image sélectionnée ou du graphique, pivotant autour d'une ligne imaginaire tracée de haut en bas de l'objet.
>
> **Vertical** Permute les bords supérieur et inférieur de l'image sélectionnée ou du graphique, pivotant autour d'une ligne imaginaire tracée horizontalement.

Rotate Ouverture d'une liste que vous pouvez utiliser pour faire pivoter l'image sélectionnée ou du graphique.

> **By...** Incliner l'image sélectionnée ou le graphique par le nombre de degrés que vous spécifiez.
>
> **90° Right** Incliner l'image sélectionnée ou graphique de 90 degrés vers la droite (sens horaire).
>
> **90° Left** Incliner l'image sélectionnée ou graphique de 90 degrés vers la gauche (sens antihoraire).
>
> **180°** Incliner l'image sélectionnée ou le graphique à 180 degrés.

Reshape Graphic Permet de remodeler interactivement le polygone sélectionné ou le graphique de la courbe.

Align Selected Controls Ouverture d'une liste que vous pouvez utiliser pour aligner des objets. Cet item est désactivé si aucun objet ou un seul objet est sélectionné.

> **Left** Déplace les contrôles sélectionnés de sorte que leurs bords gauche s'alignent avec le bord gauche du premier contrôle sélectionné.
>
> **Right** Déplace les contrôles sélectionnés de sorte que leurs bords droit s'alignent sur le bord droit de la première commande sélectionnée.
>
> **Top** Déplace les contrôles sélectionnés de sorte que leurs bords supérieurs s'alignent avec le bord supérieur du premier élément de contrôle sélectionné.
>
> **Bottom** Déplace les contrôles sélectionnés de sorte que leurs bords inférieurs s'alignent avec le bord inférieur du premier élément de contrôle sélectionné.
>
> **Make Widths Equal** Redimensionne les contrôles sélectionnés de telle sorte que la largeur de chacun est égale à la largeur du premier élément de contrôle sélectionné.

Make Heights Equal Redimensionne les contrôles sélectionnés de telle sorte que la hauteur de chacun est égale à la hauteur du premier élément de contrôle sélectionné.

Send to Back Déplace les objets sélectionnés derrière tous les autres objets de la carte. Cet item est désactivé si aucun objet n'est sélectionné.

Move Backward Déplace les objets sélectionnés en arrière d'une couche.

Move Forward Déplace les objets sélectionnés en avant une couche.

Bring to Front Déplace les objets sélectionnés en avant de tous les autres objets de la carte.

3.1.5 Le Menu Texte

Le menu texte contient les commandes pour changer l'apparence du texte.

Text Style Options Applique ou supprime le style choisi du texte sélectionné ou de l'objet sélectionné. Ces éléments sont désactivés si rien n'est sélectionné.

Link Transforme le texte sélectionné, ou tout le texte dans l'objet sélectionné en lien. Le lien texte a des propriétés spéciales dans LiveCode, par exemple texte lié recevra un message **LinkClicked** quand on clique dessus. Pour plus de détails, voir la section sur l'objet Champ.

Subscript Modifie le texte sélectionné en dessous de la ligne standard et le rend plus petit.

Superscript Modifie le texte sélectionné au-dessus de la ligne standard et le rend plus petit.

Font Ouverture d'une liste que vous pouvez utiliser pour changer le type la police utilisée pour le texte ou les objets sélectionnés.

Use Owner's Font Impose le type de police du propriétaire de l'objet (si un objet est sélectionné) ou de l'objet dans lequel le texte est (si du texte est sélectionné) pour être utilisé, en supprimant toute type de police spécifique pour le texte ou les objets sélectionnés. Pour plus d'informations sur l'héritage de la police, voir la section sur la hiérarchie des messages.

Size Ouverture d'une liste que vous pouvez utiliser pour changer la taille de la police utilisée pour le texte ou les objets sélectionnés.

Use Owner's Size Impose la taille de la police du propriétaire de l'objet (si un objet est sélectionné) ou de l'objet dans lequel le texte est (si du texte est sélectionné) pour être utilisé, en supprimant toute taille de caractères particulière pour le texte ou les objets sélectionnés. Pour plus d'informations sur l'héritage de la police, voir la section sur la hiérarchie des messages.

Color Ouverture d'une liste que vous pouvez utiliser pour changer la couleur de la police utilisée pour le texte ou les objets sélectionnés. (Cet item est désactivé si rien n'est sélectionné.)

Use Owner's Color Impose la couleur de la police du propriétaire de l'objet (si un objet est sélectionné) ou de l'objet dans lequel le texte est (si du texte est sélectionné) pour être utilisé, en supprimant toute couleur de police spécifique pour le texte ou les objets sélectionnés. Pour plus d'informations sur l'héritage de la police, voir la section sur la hiérarchie des messages.

Pen Color Modifie le texte sélectionné, ou le texte utilisé dans les objets sélectionnés, en utilisant le réglage actuel du crayon de couleur (utilisé pour dessiner des graphiques à partir de la palette d'outils principale).

Align Ouverture d'une liste que vous pouvez utiliser pour modifier l'alignement du texte (justification) utilisé dans les objets sélectionnés.

3.1.6 Le Menu Développement

Le menu de développement contient les commandes pour le débogage et pour l'utilisation de piles d'outils personnalisés.

LiveCode Online Ouvre la fenêtre LiveCode en ligne, vous permettant de partager vos piles avec la

communauté LiveCode et de télécharger les piles d'autres personnes.

Object Library Affiche la fenêtre Object Library, qui stocke des objets pré-scriptés, que vous pouvez copier dans la pile actuelle pour emploi.

Image Library Affiche la fenêtre de ***Image Library***, qui affiche les images que vous pouvez référencer ou copier dans la pile actuelle pour emploi. Vous pouvez utiliser ***Image Library*** pour afficher toutes les icônes et curseurs qui viennent avec LiveCode, toutes les images dans la pile actuelle, ou des bibliothèques d'images que vous créez. Pour plus de détails, voir la section ***Object*** et ***Image Library***.

Plugins Ouverture d'une liste que vous pouvez utiliser pour ouvrir les piles d'outils personnalisés stockées dans le dossier ***Plugins***.

Plugin Settings Personnalise les messages qui sont pilotés par les piles d'outils personnalisés stockés dans le dossier ***Plugins***.

Script Debug Mode Si cette option est cochée, le débogueur est activé : la fenêtre débogueur s'affiche lorsqu'un point d'arrêt est rencontré au cours de l'exécution du script, et vous pourrez accéder au débogueur lorsqu'une erreur d'exécution se produit. Si elle ne l'est pas, le débogueur est désactivé. Pour plus de détails, voir la section sur le débogage.

Clear All Breakpoints Supprime tous les points d'arrêt que vous avez utilisé pour baliser l'éditeur de code, dans toutes les piles ouvertes. Note : Cette option n'affecte pas les points d'arrêt définis avec la commande de point d'arrêt.

Message Watcher Ouverture de l'Observateur de Message, utilisé pour consulter et suivre les messages, les appels de fonction, les appels **getProp** et les déclencheurs **setProp**, tels qu'ils sont envoyés aux objets.

Variable Watcher Ouvre la fenêtre l'Observateur de Variable, qui vous permet de garder une trace de la valeur des variables au cours du débogage. Lorsque le débogueur n'est pas ouvert, l'observateur de variables montre la valeur des variables globales.

Attention : cette fonction provoque dans vos scripts un échec silencieux au lieu de donner un message d'erreur dans le LiveCode lorsqu'un événement rencontre une erreur.

Suppress Errors Empêche l'affichage de la fenêtre d'erreur lorsque LiveCode rencontre une erreur de script.

Suppress Messages Empêche les messages du système (comme **openCard** et **closeCard**) d'être envoyés au cours de la navigation normale. Cette option permet également d'annuler les messages actuellement en cours.

Astuce : Pour suspendre également les bibliothèques LiveCode, maintenez la touche Maj enfoncée tout en choisissant cette option.

Suspend Development Tools Cache les menus de LiveCode, palettes et autres parties de l'environnement de développement, de sorte que que vous pouvez obtenir un aperçu de votre application et de son comportement de façon autonome, en dehors de l'environnement de développement.

3.1.7 Le Menu View

Le menu Affichage contient les commandes pour se déplacer autour de la pile actuelle et pour afficher ou masquer les outils de développement.

Go First Va à la première carte de la pile en cours.

Go Prev Remonte à la carte précédente dans la pile actuelle.

Go Next Avance à la carte suivante de la pile actuelle.

Go Last Va à la dernière carte de la pile actuelle.

Go Recent Remonte à la carte où vous étiez avant de naviguer à la carte actuelle.

Toolbar Text Cache ou affiche les étiquettes de texte dans la barre d'outils en haut de l'écran. Pour masquer la barre d'outils complètement, décochez à la fois cet item et ***Toolbar Icons***.

Toolbar Icons Cache ou affiche les icônes de la barre d'outils en haut de l'écran.

Palettes Cache ou affiche toutes les palettes de LiveCode ouvertes.

Rulers Cache ou affiche une règle sur les bords gauche et en bas de chaque pile ouverte.

Grid Si cette option est cochée le déplacement et le redimensionnement des objets est limité par une grille de pixels. Si elle ne l'est pas, vous pouvez déplacer et redimensionner des objets à n'importe quel endroit. Vous pouvez modifier l'espacement de la grille dans les **Préférences**.

Backdrop Cache ou affiche une toile de fond pleine ou à motifs derrière les fenêtres de LiveCode.

LiveCode UI Elements in Lists Si cette option est cochée, les éléments de l'environnement de développement de LiveCode apparaissent dans les listes : par exemple, les piles de l'environnement de développement apparaissent dans le navigateur de l'application et les propriétés personnalisées de LiveCode apparaissent dans le volet de l'inspecteur de propriété Propriétés personnalisées. Si elle ne l'est pas, les éléments de l'environnement de développement de LiveCode n'apparaissent pas dans ces listes.

Look and Feel Ouverture d'une liste que vous pouvez utiliser pour modifier l'apparence des contrôles afin de prévisualiser l'apparence de votre application sur d'autres plateformes.

Show Invisible Objects Si cette option est cochée, les objets dont la propriété visible est définie sur désactivé sont affichés. Si elle ne l'est pas, les objets dont la propriété visible est défini sur désactivé restent cachés.

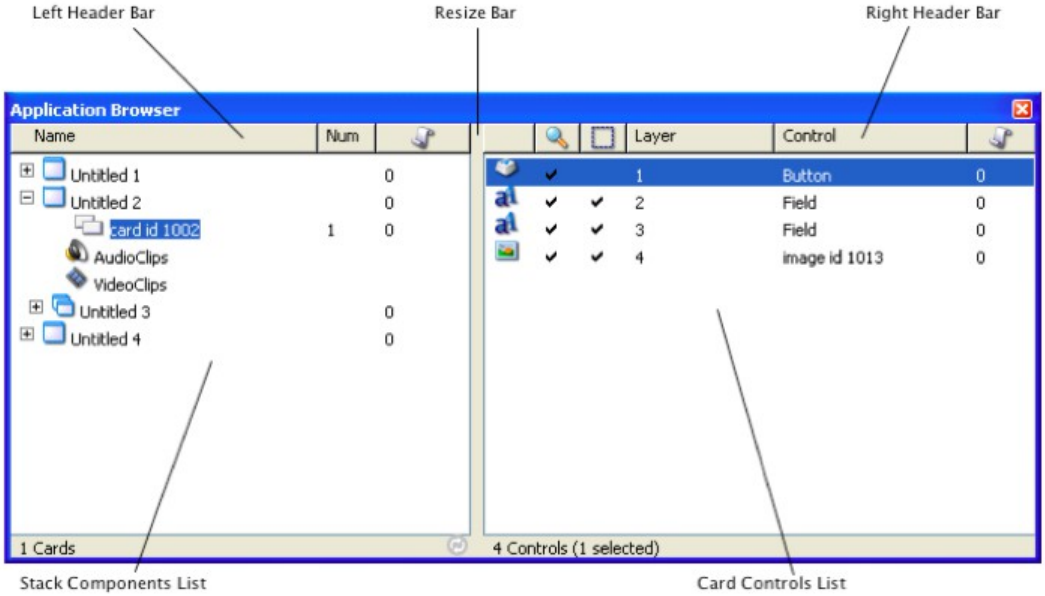
3.1.8 Le Menu Window

Le menu ***Window*** contient les noms des fenêtres des piles ouvertes.

Send Window To Déplace la fenêtre de la pile du premier plan vers l'arrière plan et ramène la seconde fenêtre vers l'avant. (Cet item est désactivé si une seule de pile est ouverte ou la fenêtre active n'est pas une pile.)

3.2 Le Navigateur d'Application (ApplicationBrowser)

Le navigateur d'application contient une liste de toutes les piles ouvertes, les cartes dans chaque pile, et les contrôles sur chaque carte. Il vous permet de naviguer vers n'importe quelle carte, ouvrir ou fermer une pile, sélectionner, ouvrez l'inspecteur de propriétés pour, ou modifier le script d'un objet quelconque. Vous pouvez accéder au navigateur de l'application choisissant **Tools / Application Browser**.



Application Browser : main window

Left header bar Affiche le nom des piles et de leurs cartes associées, ainsi que le numéro des objets et le nombre de lignes de script pour pour cette carte ou cette pile. Cliquer dans cette zone va trier les cartes par l'en-tête de colonne choisi. Un clic droit vous permet de personnaliser les colonnes sur l'écran, ajouter éventuellement des colonnes pour afficher le numéro de l'objet ou de la propriété marquée de la carte, ou vous permet de réinitialiser les colonnes par défaut. Cliquer et glisser entre les en-têtes de colonnes vous permet de redimensionner les colonnes.

Stack components list Affiche toutes les piles (**mainStacks** et **subStacks**) à l'intérieur des fichiers des piles ouvertes. Cliquez sur le symbole repli à la gauche de la pile pour afficher une liste de cartes, audio clips et vidéoclips associés à la pile. Double-cliquer ira à la pile ou la carte sélectionnée, ou la lecture audio ou un clip vidéo. Faites un clic droit pour ouvrir le menu contextuel de l'objet. **Alt + double-clic** pour ouvrir l'inspecteur de de l'objet et **Control-double-clic** pour éditer le script.

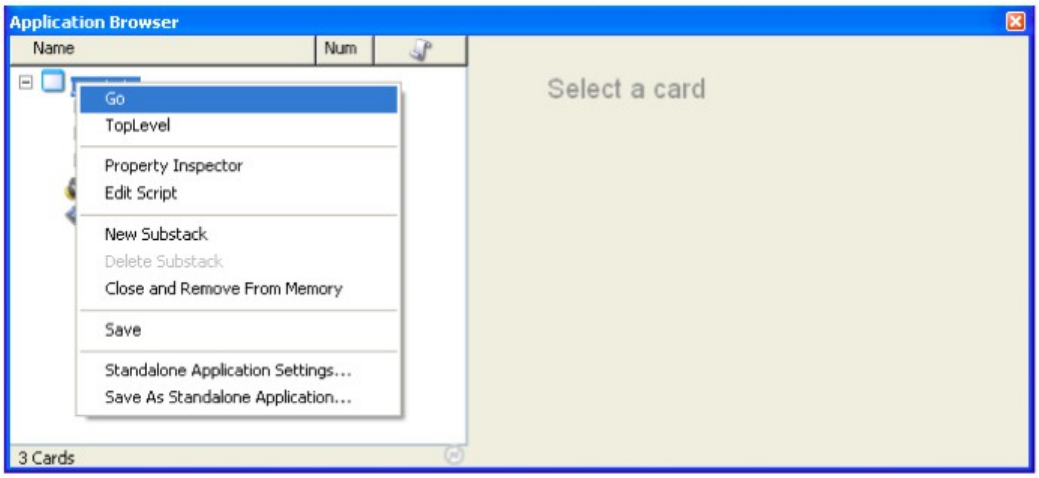
Resize bar Faire glisser pour modifier la largeur des côtés gauche et droit.

Right header bar Affiche le type de l'objet, visibilité, possibilité de sélection, couche, le nom et le nombre de lignes de script.

Cliquer dans cette zone va trier les objets par l'en-tête de colonne choisi. Un clic droit vous permet de personnaliser les colonnes sur l'écran, ajouter éventuellement des colonnes pour afficher le numéro d'objet, ou vous permet de réinitialiser les colonnes par défaut. Cliquer et glisser entre les en-têtes de colonnes vous permet de redimensionner les colonnes.

Card controls list Affiche une liste de contrôles sur la carte sélectionnée sur la gauche, ou la liste des clips audio ou vidéo. Cliquez sur un objet pour le sélectionner. Faites un clic droit pour ouvrir le menu contextuel de l'objet. Un **double-clic** pour ouvrir un inspecteur de contrôle ou jouer un **audioClip** ou un **videoClip**. **Alt + double-clic** pour ouvrir l'inspecteur d'un **audioClip** ou d'un **videoClip**. Un **contrôle clic** pour sélectionner plusieurs objets. Un **contrôle double-clic** pour modifier le script. Pour actualiser la liste des commandes en cours d'affichage dans la colonne de droite, faites un clic droit sur un objet, puis choisissez **Actualise**.

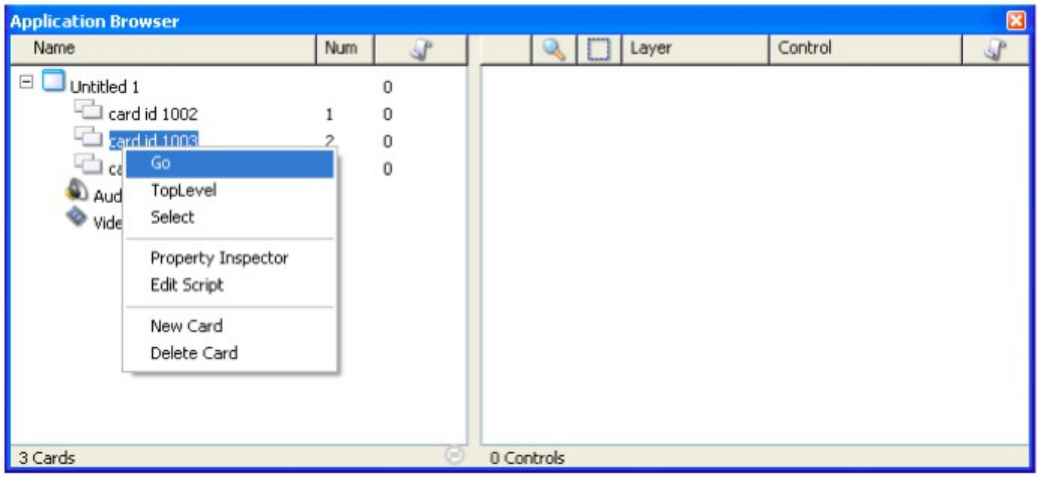
Astuce : Pour sélectionner un objet en tapant son nom, cliquez soit dans la colonne de gauche ou la colonne de droite, puis commencez à taper.



Application Browser : pile menu contextuel

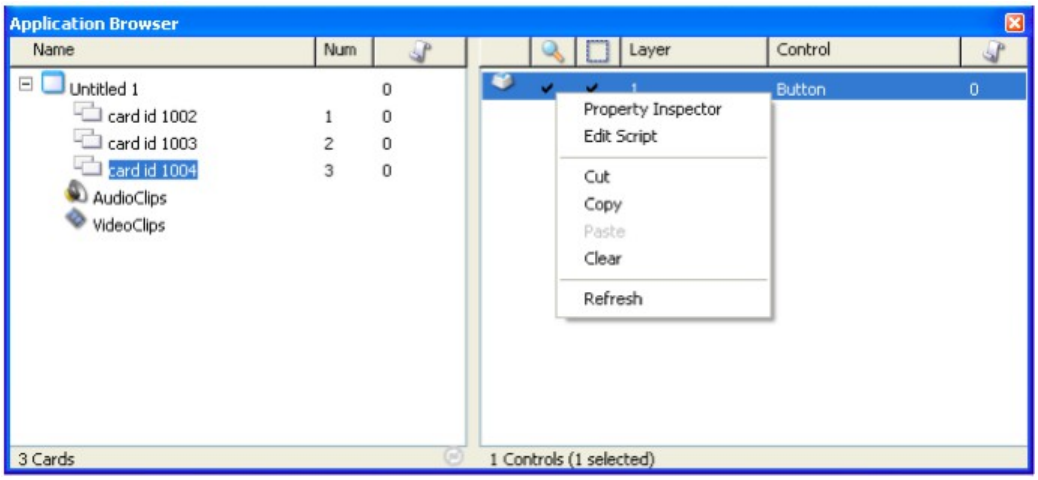
Go Aller à cette pile.
Toplevel Ramenez cette pile au premier plan en mode édition.
Property Inspector Ouvrez l'Inspecteur pour cette pile.
Edit Script Modifier le script de cette pile.
New Substack Créer un une sous-pile dans le même fichier que cette pile.
Delete subStack Supprimer cet une sous-pile.

Close and Remove From Memory Fermer et supprimer les piles principales et toutes les sous-piles dans le fichier de la mémoire.
Save Enregistrer toutes les piles dans ce fichier de pile.
Standalone Application Settings Ouvrez les options des paramètres pour applications autonomes.
Save as Standalone Application Enregistrez ce fichier pile comme une application autonome.



Application Browser : Carte menu contextuel

Go Aller vers cette carte
Toplevel Amener la pile au premier plan en mode édition, puis accédez à cette carte
Select Selectionner cette carte
Property Inspector Ouvrez la fenêtre Inspecteur pour cette carte
Edit script Modifier le script de cette carte
New Card: Créer une nouvelle carte
Delete Card: Effacer cette carte



Application Browser : Menu commande contextuelle
Property Inspector Ouvrez la fenêtre Inspecteur pour cette commande
Edit script Modifier le script de cette carte
Refresh Mettre à jour la liste des contrôles

3.3 L'inspecteur de propriétés (The Properties Inspector)

L'inspecteur de propriétés vous permet de visualiser et d'éditer les propriétés de n'importe quel objet sélectionné. Les propriétés contrôlent l'aspect d'un objet et certains aspects du comportement d'un objet. L'inspecteur peut être consulté en cliquant deux fois sur un objet sélectionné, à partir de la barre d'outils, dans le menu de l'objet et depuis les menus contextuels.

Lock icon L'icône de verrouillage verrouille l'inspecteur de l'objet en cours d'inspection. Une fois verrouillé, l'inspecteur ne sera pas mis à jour lorsque vous modifiez la sélection ou passer en mode exécution en choisissant **Browse Tool**. Cela vous permet d'utiliser un inspecteur verrouillé pour modifier les propriétés d'un objet tout en interagissant avec votre projet. En sélectionnant un autre objet et en choisissant **Inspect** cela va créer un autre inspecteur pour inspecter le nouvel objet. Vous pouvez utiliser cette fonction pour comparer les propriétés des deux objets.

Pane Selector Ce menu vous permet d'accéder à chacun des différents volets au sein de l'inspecteur pour un objet donné.

Action menu Utilisez le menu Action pour sélectionner un autre objet, modifier le profil, modifiez le script ou envoyez un message à l'objet en cours d'inspection, ou pour verrouiller l'inspecteur.

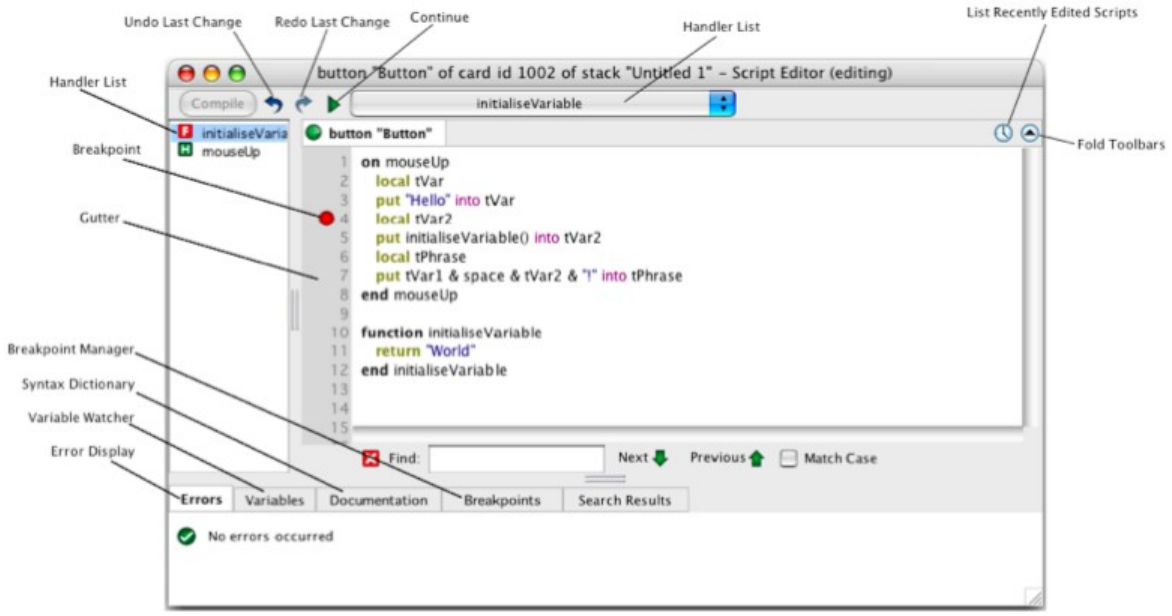
Property text Modifier le contenu d'une propriété en tapant dans la zone de texte à l'intérieur de l'inspecteur. Appuyez sur **Entrée** pour affecter la propriété et laisser l'option en cours active.. Appuyez sur **Tab** pour définir la propriété et de passer à la prochaine propriété de texte modifiable.

Property description Ce texte décrit la propriété de l'objet. Par défaut, ce texte est une description de la propriété proche de l'anglais. Cependant, vous pouvez changer cela en activant **Name of the LiveCode Property** dans **Preferences**. Vous préférerez peut-être voir les noms de propriété de LiveCode si vous écrivez des scripts qui définissent les propriétés des objets.

Important : Pour la documentation sur chaque propriété d'un objet particulier, passez la souris sur de l'objet jusqu'à ce que vous voyez la propriété du script équivalent. Ensuite, regardez ce terme dans le Dictionnaire du LiveCode.

3.4 L'Editeur de code (The Code Editor)

L'éditeur de code au sein de LiveCode a été spécialement conçu pour le codage de LiveCode. Il inclut des fonctionnalités pour aider à rendre le code plus compréhensible. Il s'agit notamment de l'indentation du code et la coloration syntaxique codée, ainsi que d'autres outils intégrés tels qu'un débogueur et Dictionnaire de syntaxe. Vous pouvez accéder à l'éditeur de code pour un objet en sélectionnant l'objet puis en choisissant le script dans la barre d'outils. L'éditeur de code est également disponible dans le menu de l'objet, et à partir d'un certain nombre d'autres menus contextuels détaillés ailleurs dans ce guide. Chacune des composantes de l'éditeur de code sont décrites ci-dessous.



Main Script Area Affichez et modifiez des scripts dans cette zone. Pour plus de détails sur la façon d'écrire un script, consultez la section sur le codage dans LiveCode. L'éditeur de code colorise et formate automatiquement les scripts. Lors de la saisie, appuyez sur **Tab** pour formater manuellement script. Appuyez sur **Entrée** (sur le pavé numérique) pour compiler le script. Appuyez sur **Entrée** à nouveau pour fermer l'éditeur de code.

Remarque : Il n'est pas nécessaire de fermer l'éditeur de code pour exécuter un script, tout ce qui est exigé c'est que le script soit validé en utilisant le bouton **Apply**.

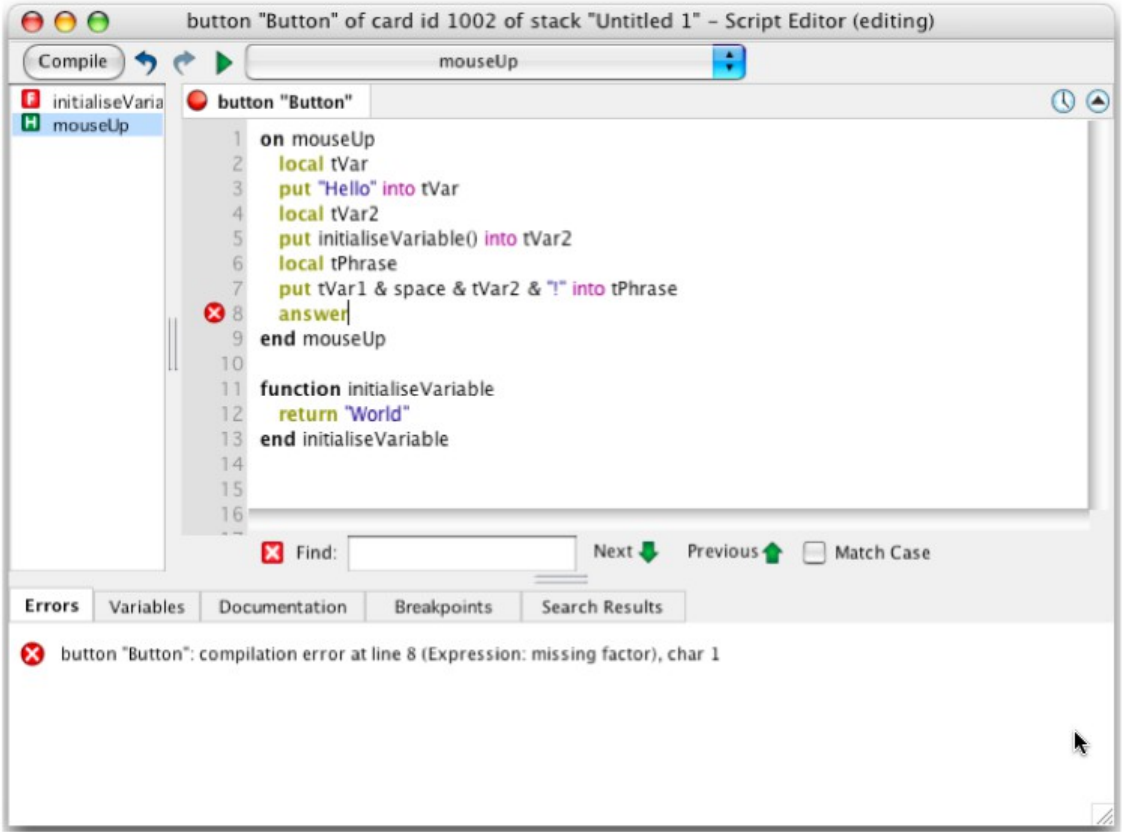
Breakpoints Area Cliquez à côté d'une ligne du script pour créer ou supprimer un point d'arrêt. Un point d'arrêt spécifie l'endroit où l'exécution du script s'interrompt et le débogueur est lancé. Pour plus de détails, voir la section sur le débogage.

Apply Button Compile le script instantanément. Les erreurs de syntaxe apparaissent sur l'écran d'erreur et le script ne compilera pas. Un script compilé n'est pas enregistré jusqu'à ce que la pile qui le contient ne soit enregistrée.

Astuce : Appuyez sur **Entrée** (sur le pavé numérique) pour activer le bouton **Apply**. Appuyez sur **Entrée** à nouveau pour fermer l'éditeur de code.

Important : Notez qu'aucun des gestionnaires de messages dans le script ne sera accessible si il ya une erreur de script lors de la compilation, car aucun des scripts n'aura été compilé.

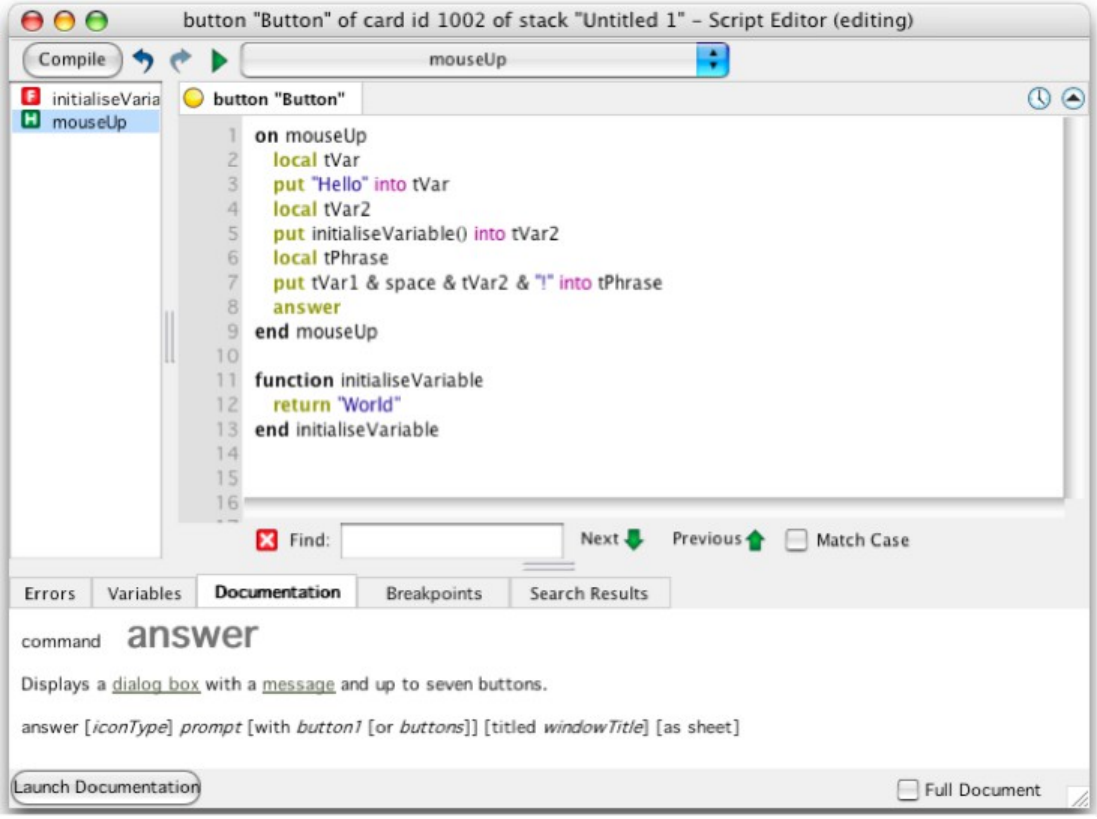
L'observateur d'erreur (The Error Watcher) Les erreurs de compilation et d'exécution apparaîtront ici. Double-cliquez sur l'icône à côté des détails de l'erreur pour mettre en évidence la ligne où l'erreur s'est produite dans le script (le cas échéant).



Handler List La liste des gestionnaires affiche toutes les fonctions, les commandes et les gestionnaires qui font partie du script en cours. Lorsqu'un nom de gestionnaire est cliqué, le script va afficher son contenu.

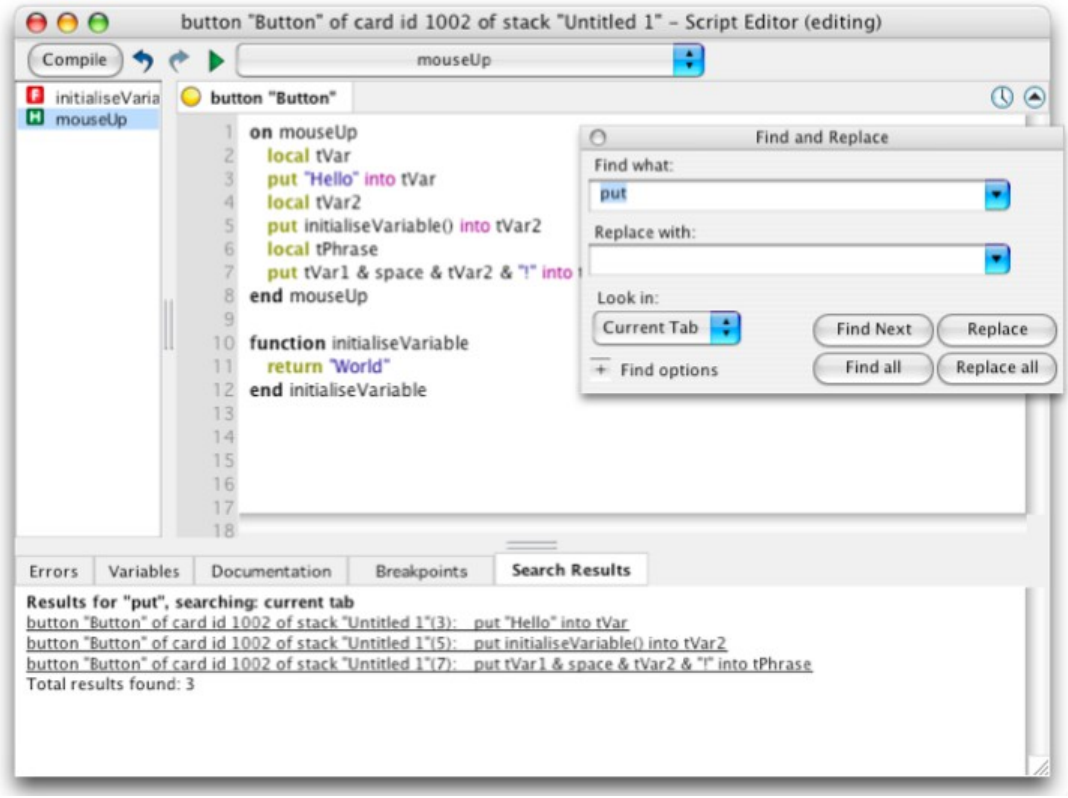
Documentation

L'éditeur de code dispose également d'un dictionnaire de syntaxe intégré. Lorsque cet onglet est actif, un résumé de l'entrée de dictionnaire pour le mot-clé que vous êtes en train de taper sera affiché. Les entrées complètes du dictionnaire peuvent être consultées en cliquant sur **Launch Documentation**. Sinon, vous pouvez choisir d'afficher l'ensemble de l'entrée active en cochant la case **Full Document** au bas de la fenêtre de documentation.



Search Results

Lorsque vous effectuez une opération **find all** les résultats seront affichés dans cet onglet.



Script Tabs

Lorsque plusieurs scripts sont ouverts, ils apparaissent comme des onglets dans l'éditeur de code.

Un clic droit sur les onglets affiche un menu qui permet à un onglet d'être déplacé dans une nouvelle fenêtre, fermé ou gardé en fermant tous les autres onglets.

Cliquer un onglet pour faire apparaître le script qu'il représente.

3.5 Le Débogueur (The Debugger)

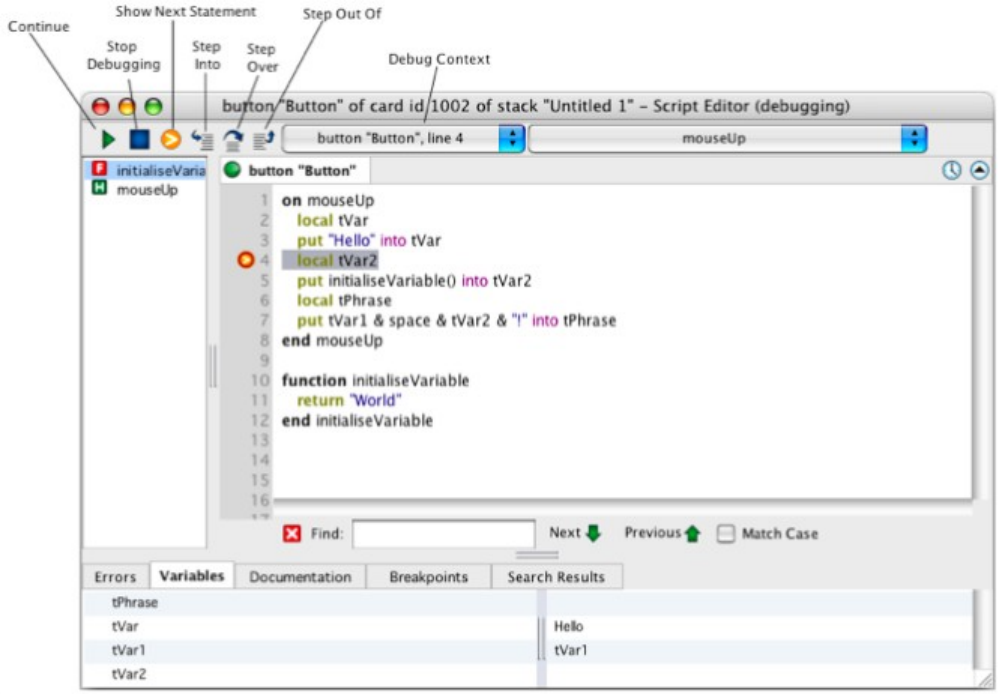
Le débogueur vous permet de suivre les bugs et de comprendre le code, en vous permettant de mettre en pause l'exécution de vos programmes ou de les parcourir ligne par ligne.

Vous provoquez l'exécution d'une pause en fixant des points d'arrêt.

Lorsque vous exécutez le script, l'exécution s'interrompt à un point d'arrêt et vous serez en mesure

- d'observer ou modifier les valeurs des variables,
- de poursuivre l'exécution ou
- de créer un nouveau point d'arrêt plus tard dans le code.

Chaque caractéristique du débogueur est décrite ci-dessous:



Continue En mode **debug**, le bouton **Continue** va commencer l'exécution du code depuis la position actuelle. Le programme se poursuivra jusqu'à ce que la fin soit atteinte ou qu'un autre point d'arrêt soit trouvé. Lorsque vous n'êtes pas en mode **debug**, le bouton **continue** peut être utilisé pour exécuter une commande (***handler***). Lorsque vous appuyez dessus, une boîte de dialogue s'affiche, vous demandant quelle commande que vous souhaitez appeler et, le cas échéant, quels sont les paramètres que vous souhaitez passer.

En cliquant sur **OK**, l'éditeur de code va appeler la commande spécifiée. Il permettra également de rappeler la commande pour la prochaine fois. Pour changer la commande appelée ultérieurement, choisissez **Entry Point**... dans le menu **Debug**.

Stop Debugging

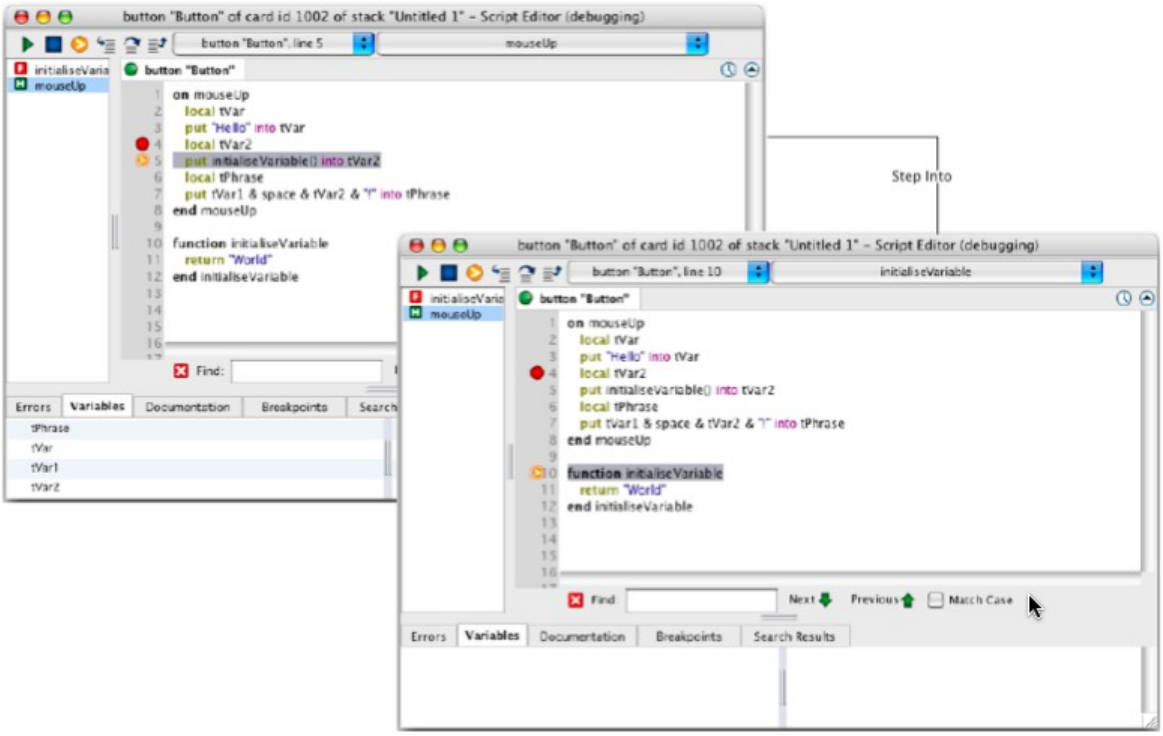
Va arrêter l'exécution au point courant et arrêter le débogage. Vous serez alors en mesure de modifier le script.

Show Next Statement

Cette option remettra le débogueur à l'instruction en cours d'exécution. Ceci est utile si vous avez changé d'onglet ou fait défiler le code pendant le débogage.

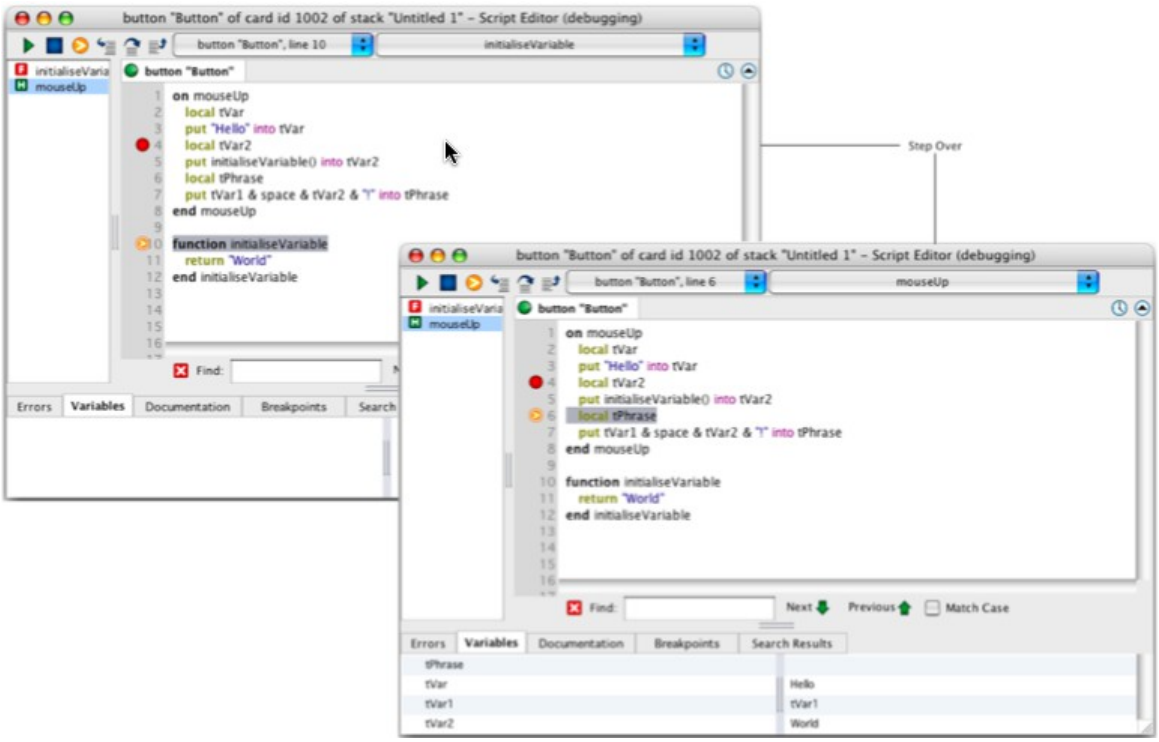
Step Into Next Statement

Utilisez **Step Into** pour exécuter l'instruction suivante. Si la déclaration suivante est une commande ou une fonction, **Step Into** ira à cette fonction ou une commande permettant d'exécuter ligne par ligne.



Step Over Next Statement

Utilisez **Step Over** pour exécuter un appel de commande ou d'une fonction sans progresser à travers elle ligne par ligne. Le code dans le gestionnaire fonctionnera à pleine vitesse et l'exécution s'arrêtera sur la ligne après l'appel de la commande.



Step Out

Cette fonction vous permet de sortir d'une commande ou d'une fonction déjà en **Stepped Into**. Une fois sélectionnée, le reste du gestionnaire actuel se déroulera et l'exécution s'arrêtera sur la ligne après l'appel de la commande. Ceci est utile pour éviter d'exécuter une longue commande ou une ligne de fonction en ligne lorsque vous avez identifié que l'erreur que vous recherchez n'est pas dans cette commande ou fonction.

Debug Context

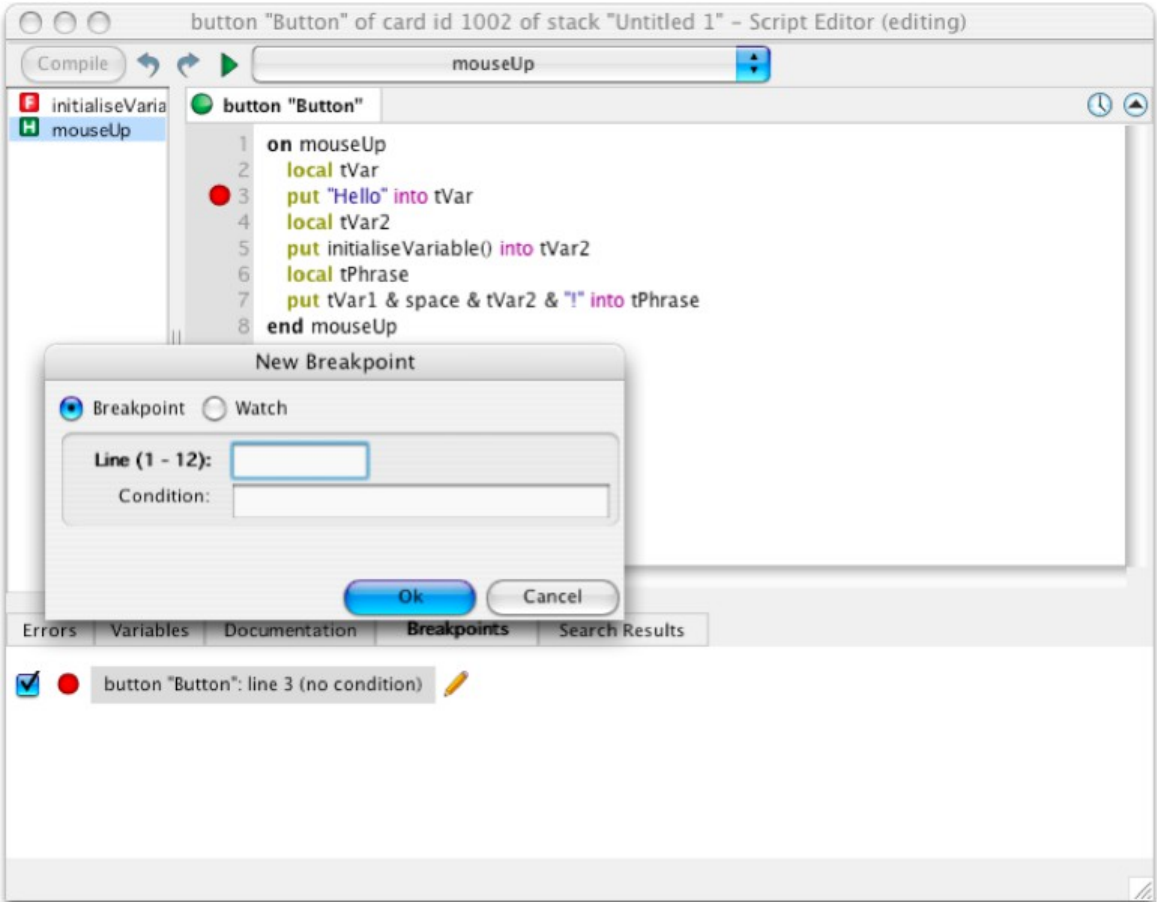
Le contexte de débogage vous montre le chemin d'exécution jusqu'à l'instruction en cours que vous avez suspendu dans le débogueur. Cela vous permet de déterminer où vous êtes dans votre programme et quelles commandes ont appelé la commande actuelle où vous êtes. Changer le contexte de débogage vous permet de visualiser les variables des différentes parties du programme. Utilisez cette fonction pour trouver un appel de commande erronée ou un appel dans lequel des paramètres erronés ont été utilisés.

Variable Watcher

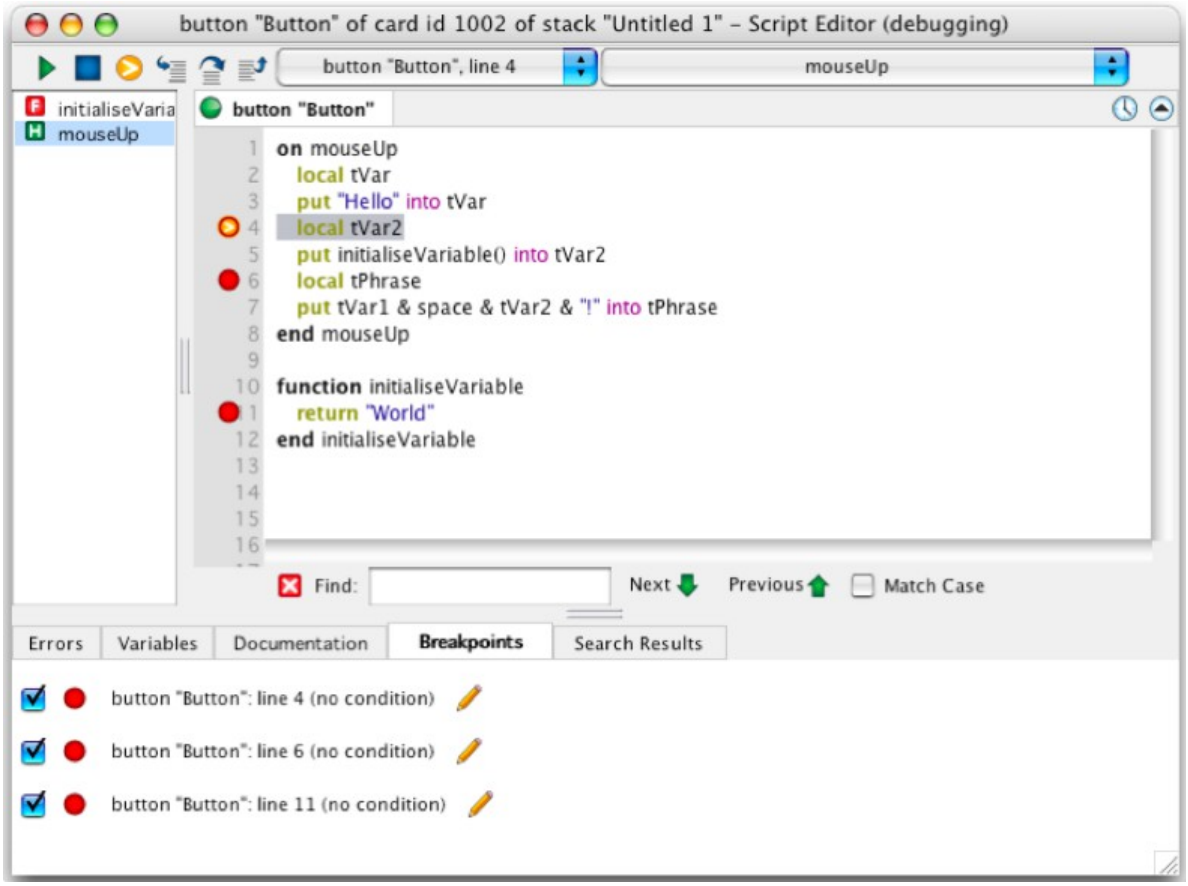
L'onglet observateur de variable vous permet d'examiner la valeur des variables au sein de votre programme pendant son exécution. Quand vous entrez dans votre code ces valeurs elles seront mises à jour. Le nom de la variable est affichée dans la colonne de gauche et sa valeur à côté sur la droite. Si la valeur d'une variable est trop grande pour être affichée dans le volet il aura une loupe à côté de lui. Cliquer dessus fera apparaître une fenêtre de visualisation qui permet de corriger la valeur de la variable à modifier. Les variables peuvent aussi être modifiés en double-cliquant sur leur valeur dans l'observateur variable.

Astuce : Vous pouvez voir la valeur d'une variable en déplaçant la souris sur le nom des variables dans le champ du script pendant le débogage.

Utilisez des points d'arrêt pour spécifier où suspendre l'exécution dans le débogueur.

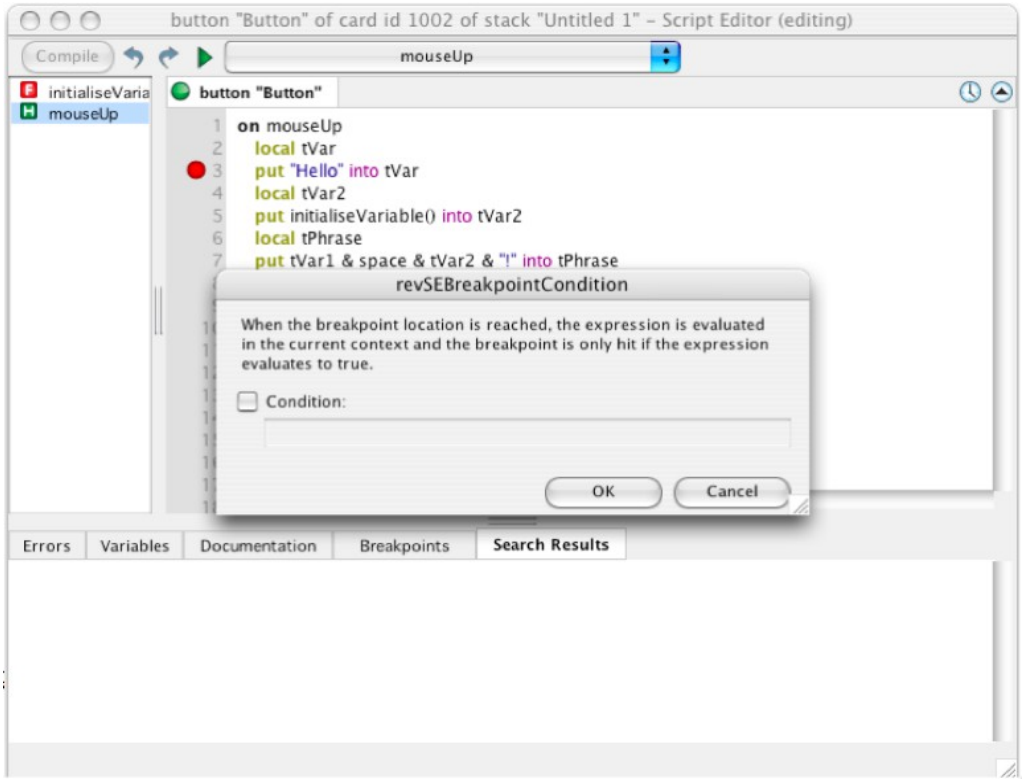


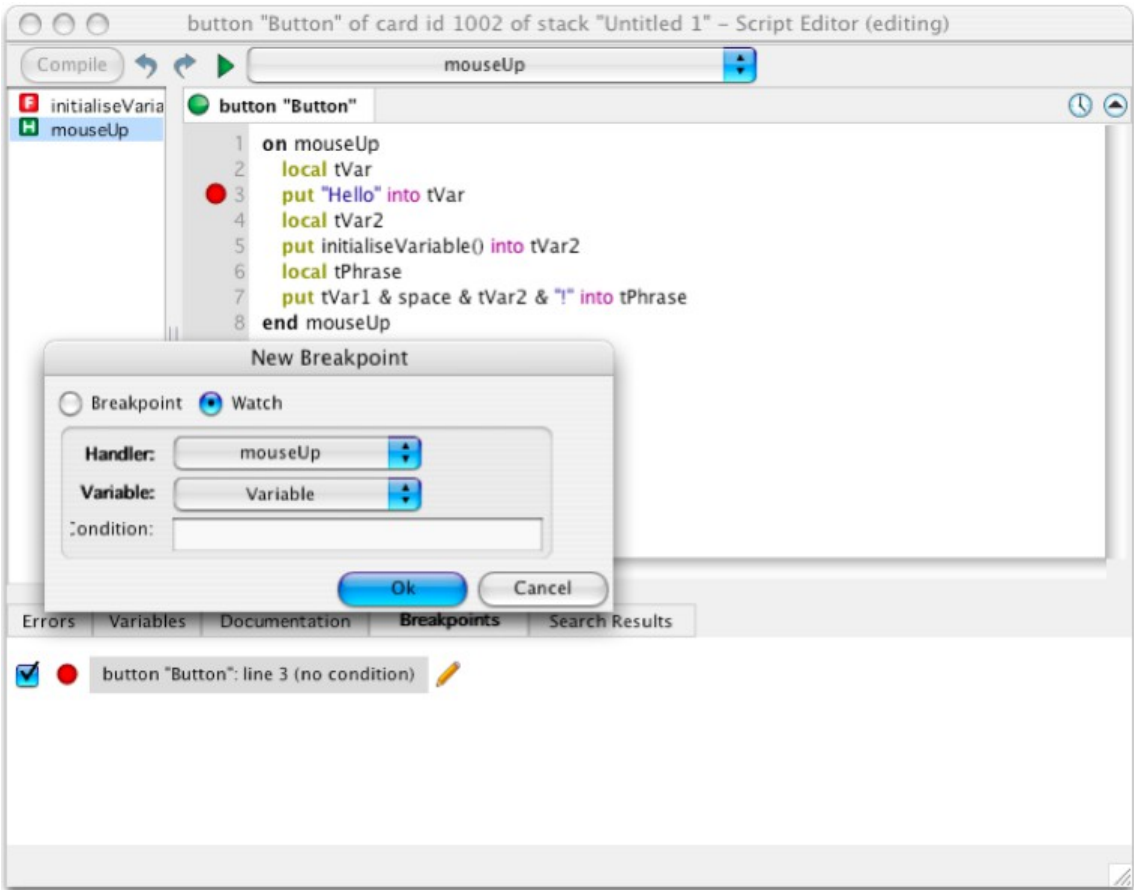
Vous définissez des points d'arrêt en cliquant sur le côté, ou par un **clic droit** dans le Gestionnaire des points d'arrêt au bas de l'éditeur de code et sélectionnez **New Breakpoint**.
Chaque point d'arrêt est associé à une ligne de code, et l'exécution est interrompue lorsque cette ligne est atteinte. Alternativement, une condition peut être affectée à l'arrêt.
Dans ce cas, l'exécution sera interrompue seulement si la condition est vraie lorsque la ligne est atteinte.



Pour voir une liste de tous les points d'arrêt dans une pile, y compris le script dans lequel ils se trouvent, quelle ligne sont surlignées, et si il y a une condition attachée, cliquez sur le Gestionnaire des points d'arrêt au bas de l'éditeur de code.
La case à cocher à gauche de chaque point d'arrêt peut être utilisée pour activer ou désactiver le point d'arrêt.
Double-cliquez sur un point d'arrêt pour aller à cette ligne dans le script de l'objet associé.
Pour modifier un point d'arrêt, y compris l'ajout d'une condition ou changer le numéro de la ligne, cliquez sur le crayon.

Ceci peut également être effectuée par un **clic droit** sur le **point d'arrêt**, que ce soit dans le Gestionnaire des points d'arrêt, ou sur le côté de l'éditeur de code.





Le gestionnaire de point d'arrêt vous permet également de définir des alertes. Elles sont associées à des variables plutôt que des lignes de code et mettront en pause l'exécution lorsque la valeur de la variable surveillée change. Si l'alerte a une condition attachée, l'exécution sera interrompue seulement si, et lorsque la valeur de la variable est modifiée, la condition est vraie. Pour ajouter une alerte, faites un clic droit dans le

gestionnaire de point d'arrêt et choisissez **New Breakpoint**.

3.6 La Barre de menu de l'Editeur de Code

La barre de menu de l'éditeur de code contient les commandes pour vous aider à modifier les scripts.

3.6.1 Le Menu Fichier

Le menu **File** contient les commandes pour imprimer un script ou fermer l'éditeur de code, avec ou sans enregistrement des modifications dans le script.

Apply Compile le script en cours.
Save Sauve la pile en cours.
Close Ferme le script en cours
Print Imprime le script en cours

3.6.2 Le menu Edition

En plus des commandes standard pour sélectionner, couper, copier et coller du texte, le menu **Edition** contient les commandes suivantes spécifiques :

Revert Prend toutes les modifications qui n'ont pas été compilées et les supprime à partir du script en cours.

Comment Place un caractère délimitant un commentaire au début du texte sélectionné. Si plus d'une ligne se trouve sélectionnée, un caractère de commentaire est placé au début de chaque ligne.

Uncomment Supprime les caractères de commentaire du texte sélectionné.

Quick Find Ouvre le champ de recherche dans l'éditeur de code.

Find and Replace... Ouvre le champ trouver et remplacer de l'éditeur de code.

Go Fait apparaître une boîte de dialogue qui vous permet d'entrer un numéro de ligne à atteindre dans le script.

Variable Checking La vérification des variables oblige LiveCode à effectuer un contrôle plus strict de vos scripts. En utilisant une variable sans la déclarer préalablement, ou en utilisant une chaîne littérale sans l'enclore de guillemets provoque une erreur de compilation. Ce comportement peut être utile dans la traque de certains problèmes subtils tels que faute d'orthographe dans un nom de variable.

3.6.3 Le Menu de Débogage

Le menu **Debug** contient des commandes pour aider à à déboguer les scripts :

Show Next Surligne la prochaine ligne de code qui doit être exécutée. Ceci est utile si vous avez modifié l'onglet ou fait défiler votre script lors du débogage.

Step Into Exécute la ligne de la commande en cours de débogage. Si la ligne suivante est un appel à une autre commande, celle-ci entre dans cette commande. Cet item est désactivé si le débogueur n'est pas en cours d'exécution.

Astuce : Appuyez sur **F11** pour entrer dans la ligne suivante.

Step Over Exécute la prochaine ligne de la commande en cours de débogage. Si la ligne suivante est un appel à une autre commande, celle-ci enjambe cet appel, en sautant et en restant dans la commande en cours.

Astuce : Appuyez sue **F10** pour sauter la ligne suivante.

Run Reprend normalement l'exécution de la commande en cours de débogage, à partir de la ligne actuelle.

Astuce : Appuyez sur **F5** pour activer le bouton **Execute**.

Abort Arrête l'exécution la commande en cours de débogage.

Toggle Breakpoint Variables Ajouter le signe **/** supprime un point d'arrêt à partir de **/** à la ligne sélectionnée. Modifie l'affichage de l'onglet au bas de l'éditeur de code à l'onglet **Variable Watcher** (observateur des variables).

Breakpoints Modifie l'affichage de l'onglet au bas de l'éditeur de code pour le gestionnaire de point d'arrêt.

Entry Point Permet de définir quelle exécution du gestionnaire devrait initier le débogage d'un script.

Script Debug Mode Si cette option est cochée, le débogueur est activé : l'exécution du programme se met en pause lorsqu'un point d'arrêt survient lors du l'exécution du script et vous pouvez accéder au débogueur lorsqu'une erreur d'exécution se produit. Si elle n'est pas cochée le débogueur est désactivé. Pour plus de détails voir la section sur le débogage.

3.6.4 Le menu Commande (Handler)

Le menu **Handler** contient une liste de toutes les commandes dans le script en cours. Choisissez une commande pour aller à cette commande dans le script.

3.6.5 Le Menu Window

Le menu Fenêtre contient les noms des fenêtres d'éditeur de script ouvertes.

3.7 La Boîte de Message (The Message Box)

La boîte de message est un outil de ligne de commande qui vous permet d'exécuter des scripts ou d'effectuer des opérations d'édition automatisées. Il vous permet d'essayer de courts scénarios, de tester des parties de votre programme, offre une fenêtre de résultat commode pour le débogage et peut être utilisé pour l'édition et la définition des propriétés.

Astuce : la boîte de message est l'un des éléments les plus utiles dans LiveCode quand vous êtes débutant et voulez tester des scripts simples. Vous verrez que vous pouvez essayer de nombreux exemples de scripts dans le Guide de l'utilisateur ou dans les exemples à obtenir sur notre site web par simple copier-coller dans la boîte de message.

Le message a les modes suivants :

Single Line exécuter une seule ligne et des scripts courts

Multiple Lines exécuter des scripts multi-lignes

Global Properties voir et éditer les propriétés globales

Global Variables voir et éditer les variables globales

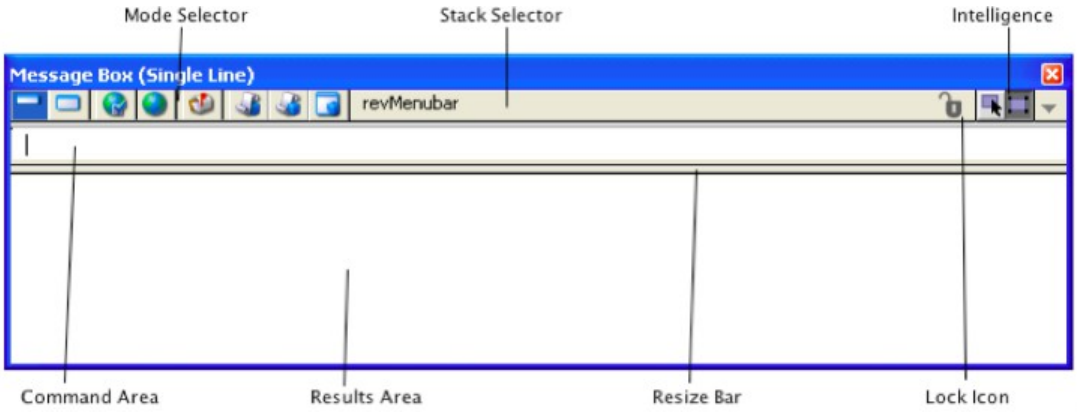
Pending Messages visualiser, supprimer et modifier des messages en attente

Front Scripts afficher et modifier des scripts du premier plan

Back Scripts afficher et modifier des scripts en arrière plan

3.7.1 Modes Simple ligne ou Multi-ligne

Utilisez les modes simple et multi-Line pour exécuter des scripts courts.



La boîte de message en mode simple ligne

Command area Tapez le code de LiveCode valide. En mode simple séparer les lignes multiples avec **;** (point-virgule) et appuyez sur **Entrée** pour exécuter. En mode **Multiple Line**, appuyez sur **Return** pour séparer les lignes et appuyez sur **Entrée** pour exécuter.

Par exemple, en mode simple ligne pour déplacer tous les contrôles sur la carte actuelle de 10 pixels vers la gauche vous devez exécuter :

```
repeat with i = 1 to the number of controls; move control i relative –10,0 ; end repeat
```

En mode multi-lignes :

```
repeat with i = 1 to the number of controls
move control i relative –10,0
end repeat
```

Taper **Control-M** pour focaliser la zone de commande et commencez à taper, si la boîte de message n'a pas le focus. Appuyez sur **Ctrl-u** lors de la saisie pour vider la zone de commande. En mode simple ligne appuyez sur la **flèche haut** pour reculer dans l'historique des scripts préalablement conclus et exécutés. Appuyez sur la **flèche bas** pour aller vers l'avant du cycle. En mode **Multiple Line**, appuyez sur **alt + flèche haut** ou **alt + flèche bas** pour aller en avant ou en arrière du cycle. Appuyez sur **Ctrl + U** pour effacer la zone de commande en cours.
En tapant le nom d'une variable globale ou d'une propriété seule, cela se traduira par l'auto-complétion de la ligne pour mettre en évidence la variable globale ou la propriété, et le contenu de cette variable globale sera inscrit dans la zone de résultats. Par exemple en tapant :

```
time
```

Sera automatiquement réécrit :

```
put the time
```

L'heure actuelle sera placée dans la zone d'affichage des résultats.

Results area Affiche le résultat après l'exécution du code dans la zone de commande. Tout script qui utilise la commande sans préciser un conteneur de destination. Toute erreur de compilation du script créé à la suite de la tentative d'exécuter du code placé dans la zone de commande. Toute erreur d'exécution créé à la suite de la tentative d'exécuter du code placé dans la zone de commande. La variable globale spéciale **msg** sera mise à jour chaque fois que quelque chose est placé dans la zone de résultats soit par script ou en éditant directement le contenu de la zone de résultats. Vous pouvez définir ou récupérer le contenu de cette variable

dans un script. Par exemple, essayez exécutant la commande suivante dans la zone de commande :

```
put the clipBoardData ; replace return with return & return in msg
```

Le résultat placé dans la zone des résultats contiendra le contenu actuel du presse-papiers avec chaque caractère de retour remplacé par un second caractère de retour.

Stack selector Sélectionner une pile à travailler. Avant d'exécuter tout script **defaultStack** sera affecté à la pile affichée dans ce menu. Par défaut, la boîte de message choisira la pile de premier plan la plus modifiable. Le menu est mis à jour chaque fois que vous changez la pile la plus en avant ou cliquez sur la boîte de message. Vous pouvez utiliser ce menu pour choisir une pile ouverte alternative. Par exemple, l'exécution du script suivant dans la zone de commande de la boîte de message d'une seule ligne

```
put the number of controls
```

placerait le nombre de contrôles au sein de la carte actuelle de la pile affichée dans le menu dans la zone de résultats.

Intelligence La boîte de message essaiera d'auto-compléter la saisie du nom d'une propriété de l'objet, en mettant le contenu de cette propriété dans la zone de résultats. Vous pouvez décider si l'auto-complétion doit tenter d'utiliser l'objet sélectionné, ou l'objet directement sous la souris.

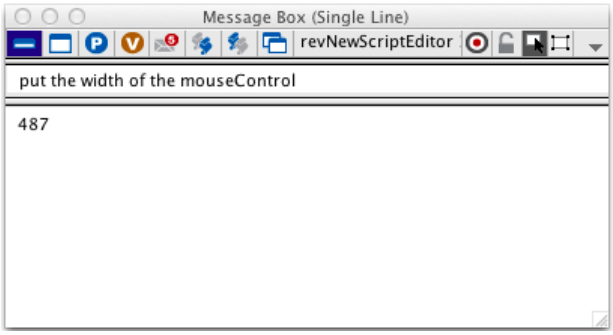
Par exemple, avec un objet sélectionné, entrer :

```
width
```

Se traduira par :

```
put the width of the selObj
```

La largeur de l'objet sélectionné sera placé dans la zone de résultats. Le **selObj** sera remplacé par **mouseControl** si vous choisissez cette option.
Dans l'exemple ci-dessus, la largeur de l'objet sous la souris est inscrit dans la zone de résultats.



Pour plus de détails sur la fonction de **selObj** ou les fonctions **mouseControl**, voir Dictionnaire de LiveCode.

Lock icon Cette option empêche le rafraîchissement automatique du sélecteur de pile, lorsque vous permutez les piles. Utilisez cette option si vous voulez exécuter plusieurs commandes sur une pile quelle que soit la pile de premier plan active.

3.7.2 Les Propriétés Globales (Global Properties)

Le mode **Global Properties** vous permet de visualiser et de modifier toutes les propriétés globales. Faites défiler pour sélectionner une propriété dans la liste de gauche et sélectionnez-la pour l'afficher ou la modifier. Taper une partie du nom de la propriété dans le champ Filtre en haut filtrera la liste des propriétés. Les modifications des propriétés globales prennent effet immédiatement. Si une propriété est «en lecture seule», cela sera indiqué lors de la sélection et l'édition, il n'y aura aucun effet. Pour plus de détails sur les propriétés globales, voir la section **onGlobal Properties**.

3.7.3 Les Variables Globales (Global Variables)

Le mode Variables Globales vous permet de visualiser et de modifier toutes les variables globales. Les variables d'environnement sont affichées en premier, suivi par d'autres variables. Faites défiler pour trouver une variable dans la liste de gauche et sélectionnez-la pour l'afficher ou la modifier. Taper une partie du nom de la variable dans le champ supérieur **Filter** filtrera la liste des variables. Les modifications prennent effet immédiatement. L'option afficher les variables **LiveCode UI** est une option avancée, présentée dans la section Modification de l'interface utilisateur de LiveCode. Pour plus de détails sur les variables globales, voir la section sur les variables globales.

3.7.4 Messages en Attente (Pending Messages)

Le mode messages en attente vous permet de visualiser tous les messages en instance basés sur la temporisation en cours. C'est identique à la propriété globale de **pending Messages**. Vous pouvez sélectionner un message en attente pour modifier son script ou l'annuler.

Astuce : Vous pouvez annuler tous les messages en attente courants avec le bouton Supprimer les messages sur la barre d'outils.

Appuyer sur rafraîchir actualise la liste avec la liste actuelle des messages en attente. Le mode **Auto Update** rafraîchit la liste en continu. Notez que la mise à jour automatique a lieu toutes les 200 millisecondes. Si vos messages arrivent plus vite ils ne pourront pas être affichés. Pour plus de détails sur **Pending Messages** , voir la section sur la messagerie basée sur le temps. L'option afficher les messages de l'interface utilisateur de LiveCode est une option avancée, présentée dans la section Edition de l'Interface Utilisateur Live Code.

3.7.5 Scripts de Premier-plan et d'Arrière-plan (Front & Back Scripts)

Ces modes listent tous les scripts actuellement disponibles en tant que bibliothèques, en avant ou en arrière du chemin de message. C'est identique aux propriétés globales **frontScripts** et **backScripts**. Sélectionnez un script supérieur ou inférieur pour le retirer ou le modifier. Pour plus de détails sur ces script, consultez la section sur l'extension du chemin de message (**Extending the Message Path**).
L'option afficher les scripts de l'interface utilisateur de LiveCode est une option avancée qui affiche toutes les bibliothèques utilisées par l'IDE. Cette question est abordée dans la section Edition de l'Interface Utilisateur.

3.7.6 Piles en cours d'Utilisation (Stacks In Use)

Ce mode est identique au mode de scripts supérieurs et inférieurs, sauf qu'il affiche la propriété globale **stacksInUse**. Vous pouvez également ajouter une pile directement en utilisant le bouton Ajouter.

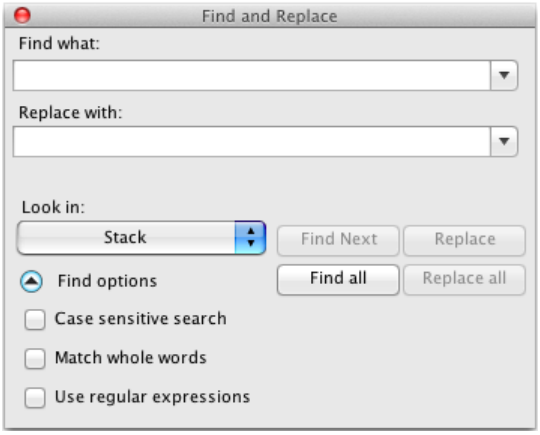
3.8 La Barre Outils (The Toolbar)

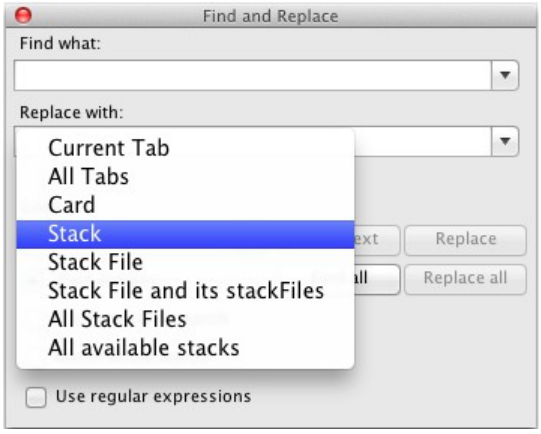
La barre d'outils principale offre un accès facile aux fonctions fréquemment utilisées.

Pour plus de détails sur ce que chaque icône de barre d'outils fait voir la section sur la barre de menus (ci-dessus).
Pour masquer et afficher du texte ou des icônes dans la barre d'outil utiliser **View / Toolbar Text** and **View / Toolbar Icons**

Pour masquer la barre d'outils complètement, décochez les deux options.

3.9 Rechercher et Remplacer (Find and Replace)





Find and Replace

La boîte de dialogue rechercher et remplacer vous permet de chercher dans l'intégralité de votre votre application, une partie de celle-ci, dans plusieurs fichiers d'un répertoire ou dans des piles spécifiées dans la propriété **stackFiles** de votre application. Vous pouvez rechercher des noms d'objets, des scripts, des champs et textes des boutons, des propriétés personnalisées et d'autres propriétés. Après avoir effectué une recherche, vous pouvez remplacer le terme recherché avec un terme de remplacement, que ce soit au sein de l'ensemble des résultats ou sur une sélection des résultats.

Find what field : Entrer le terme recherché

Replace With : Indiquer un terme à utiliser pour remplacer le terme recherché .

Look In menu : Un menu pour rechercher dans les options suivantes :

Current Tab Onglet courant

All Tabs Tous les onglets

Card La carte courante de la pile de premier plan

Stack La pile de premier plan

Stack File La pile principale et toutes les sous-piles dans le fichier de pile associé à la pile de premier plan.

Stack File and its stackFiles Comprend les piles référencées dans cette propriété **stackFiles** piles. Pour plus de détails sur la propriété **stackFiles**, voir la section sur structuration de votre application.

Important : Les fichiers de pile sont normalement attachés à la pile principale de votre application. Assurez-vous d'apporter la pile principale au premier plan avant de chercher ses fichiers de pile. Si vous voulez utiliser cette option lorsqu'un sous- pile est au premier plan, votre recherche n' inclura pas tous les fichiers de la pile.

All Stack Files Dans tous les fichiers de pile

All available stacks Dans toutes les piles disponibles

Find options Améliorer les recherches

Case Sensitive Indique que la recherche doit être sensible à la casse (par exemple "a" et "A" sont traités comme des caractères différents).

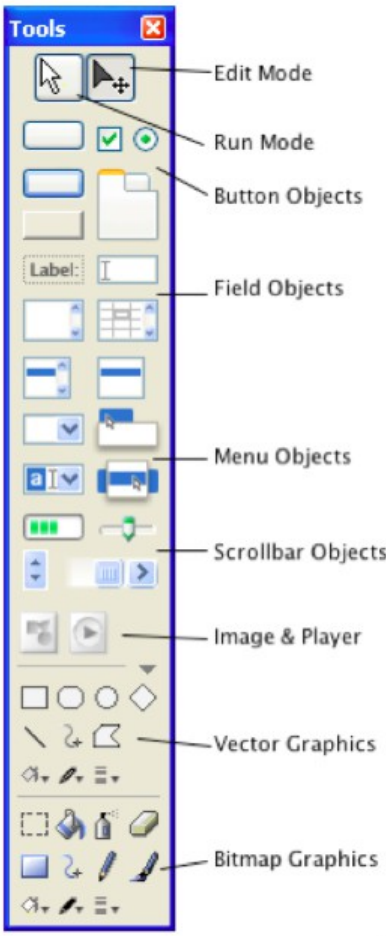
Match whole words Chercher les mots entiers.

Use regular expressions Indique que le champ de recherche contient une expression régulière au lieu de texte brut. Une expression régulière vous permet de décrire un modèle de texte pour l'adapter. Pour plus de détails sur l'utilisation des expressions régulières, consultez la section sur l'utilisation des expressions régulières ci-dessous.

Chapitre 4 Construire une Interface Utilisateur (User Interface)

L'interface utilisateur de votre application est souvent l'une de ses caractéristiques les plus importantes. Construire une interface utilisateur claire, agréable, logique et esthétique fera toute la différence pour la réussite de votre application.
Ce chapitre explique comment construire une interface utilisateur en utilisant les messages de l'interface utilisateur de LiveCode. Il vous explique comment créer et mettre des contrôles les objets à utiliser et même la façon de construire vos propres objets personnalisés.
Nous abordons ensuite brièvement quelques conseils pour une bonne conception de l'interface utilisateur.

4.1 Créer et organiser des objets



4.1.1 Créer des contrôles avec la palette des outils

La principale palette d'outils vous permet de changer entre le mode éditer ou exécuter, créer des objets et éditer des images bitmap avec les outils de peinture.

Run mode Cliquez pour entrer dans le mode d'exécution. En mode exécution, les objets reçoivent tous les messages normaux qui animent une application LiveCode. Par exemple, cliquer sur un bouton en mode exécution provoquera un message **mouseUp** à envoyer à lui-même et le script sera exécuté.

Edit mode Cliquer pour passer en mode édition. En mode édition, les objets ne reçoivent pas de messages lorsque vous cliquez sur eux, et vous pouvez les déplacer, les redimensionner ou modifier les propriétés des objets.

Pour plus de détails sur le mode exécution ou édition, consultez la section sur le mode **Editer** et **Exécuter**.

Button objects, Field objects, Menu objects, Scrollbar objects Faites glisser un de ces objets sur une pile modifiable pour créer un nouvel objet. Double-cliquez sur un objet pour créer un objet de ce type dans le centre de la pile du premier plan

Image & Player Pour plus de détails sur tous ces objets, voir la section sur chaque type d'objet plus loin dans ce chapitre.

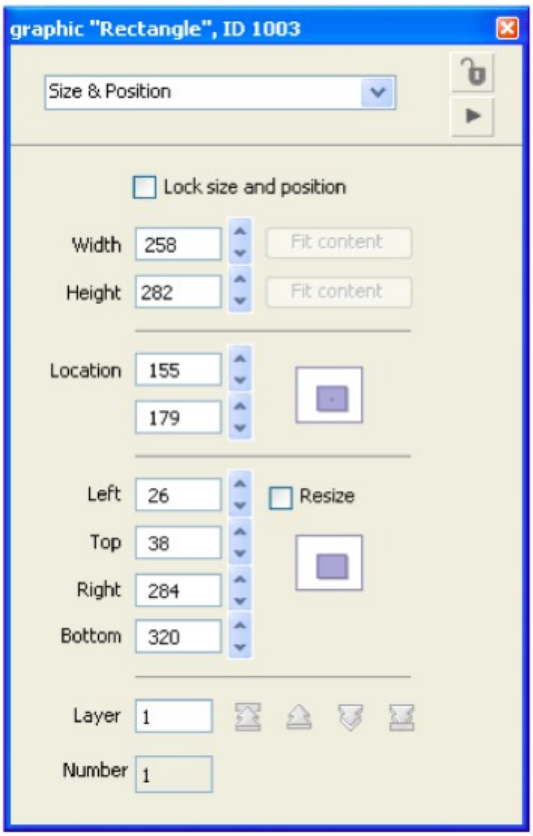
Vector Graphics Appuyez sur le triangle gris en bas à droite pour afficher ou masquer cette section de dessin vectoriel. Cliquez pour choisir le type de graphique que vous souhaitez créer. Utilisez le seau de remplissage pour choisir la couleur de remplissage, le crayon de remplissage pour choisir la couleur de la ligne, le menu de l'épaisseur de ligne de choisir l'épaisseur du trait, et le menu de la forme option pour choisir les préférences spécifiques au type de graphique choisi. Cliquez et faites glisser dans une pile modifiable pour créer le nouveau graphique.

Astuce : Vous pouvez également créer des objets en utilisant le sous-menu **New Control** dans le menu de l'objet, ou en tapant **create [objet type]** dans la boîte de message. Pour plus d'informations sur la création d'objets à l'aide de la boîte de message, voir la section sur la construction des interfaces via des scripts, plus loin dans ce chapitre.

Bitmap graphics Les outils de peinture vous permettent d'éditer des images bitmap importées dans LiveCode ou de créer la votre. Pour les utiliser, créez un objet image et les outils de dessin à l'intérieur de la zone créée, ou modifiez une image existante. Vous ne pouvez pas utiliser les outils de dessin pour modifier une image qui a son fichier de jeu de propriétés, puisque les données de l'image sont stockées à l'extérieur LiveCode. Pour plus de détails sur l'utilisation des images, voir la section sur les images plus loin dans ce chapitre et le chapitre sur le travail avec les médias.

Astuce : Pour ouvrir un sélecteur de couleur standard du système, double-cliquez sur menus surgissants au bas des sections graphiques vectoriels ou graphiques bitmap de la palette d'outils.

4.1.2 Alignement et superposition



Lock size and position Verrouille l'objet de sorte que sa taille et sa position ne peuvent être ajustées de manière interactive avec la souris en mode édition. Cela empêche aussi les images, les groupes et les lecteurs de se redimensionner automatiquement pour afficher la totalité de leur contenu chaque fois que la carte où ils se trouvent est rouverte. Pour plus de détails, voir l'entrée pour **lockLocation** dans le dictionnaire de LiveCode.

Width & Height Permet de définir la largeur et la hauteur de l'objet(s) actuellement exploité par l'inspecteur des propriétés. Les objets sont redimensionnés à partir de leur centre. Pour plus de détails, voir les propriétés de largeur et de hauteur dans le dictionnaire de LiveCode.

Fit Content Redimensionne automatiquement l'objet suffisamment grand pour pouvoir afficher tout son contenu. Dans le cas des boutons le contenu est du texte et des icônes. Pour les images, c'est la largeur et la hauteur de l'image d'origine avant toute mise à l'échelle. Pour plus de détails, consultez l'entrée **formattedWidth** et **formattedHeight** dans le dictionnaire de LiveCode.

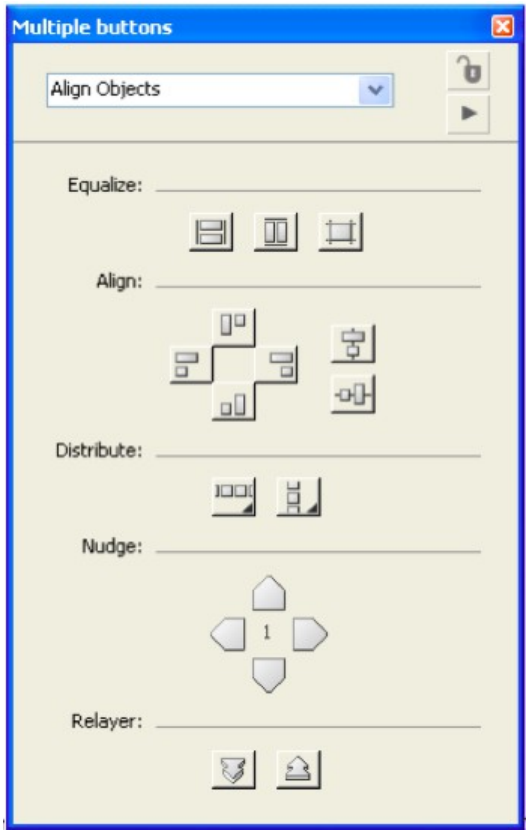
Location Définit la position des objets (le centre de l'objet) par rapport à la partie supérieure gauche de la carte.

Left, Top, Right & Bottom Définit la position de l'un des bords de l'objet.

Layer Définit le calque de l'objet. Les boutons avec des flèches vous permettent d'envoyer un objet à l'arrière, de déplacer un objet en arrière d'une couche, de placer un objet en avant d'une couche et de placer un objet au premier plan. Les calques déterminent quels objets sont affichés au dessous ou au dessous des autres, ainsi que le numéro de l'objet et de l'ordre de tabulation. Notez que vous ne pouvez pas re-calquer des objets qui sont regroupés sauf si vous êtes en mode d'édition d'arrière plan, ou avez les **relayerGroupedControls** définis sur vrai. Pour plus de détails, voir la section sur le groupe et les arrières-plans. Pour plus de détails sur l'ordre de tabulation voir la section sur le focus du clavier ci-dessous.

Inspecteur Alignement des Objets Utilisez **Align Objects Inspector** pour redimensionner des objets par rapport à d'autres objets, pour repositionner des objets et/ou pour re-calquer (changer de niveau) les objets. Pour l'ouvrir, sélectionner plusieurs objets, puis ouvrez l'inspecteur et choisissez Aligner les objets dans le menu en haut de l'inspecteur.

Le volet aligner les objets s'affiche automatiquement si vous sélectionnez plusieurs objets de types différents.



Important : L'inspecteur pour aligner les objets redimensionne les objets par rapport aux autres. Toujours sélectionner l'objet que vous souhaitez utiliser comme une référence en premier, puis sélectionner d'autres objets que vous voulez faire identiques au premier. Si vous souhaitez les répartir (distribution), sélectionnez-les dans l'ordre comme vous souhaitez qu'ils soient répartis.

Equalize Harmonisez les objets à la même largeur, hauteur ou sur le même rectangle.

Align Alignez les objets à gauche, à droite, en haut ou en bas, par leur centre horizontal ou leur centre vertical, en utilisant le premier objet sélectionné en tant que référence.

Distribute Distribue les objets avec une différence égale entre eux, en utilisant l'ordre dans lequel ils ont été sélectionnés. Du premier au dernier sélectionné, répartissez les objets de façon égale entre les bords des premiers et des derniers objets sélectionnés. **Bord à bord** distribuera les objets de sorte que leurs bords se touchent exactement. **A travers la carte** va distribuer les objets uniformément sur toute la surface de la carte.

Nudge Déplacer l'objet sélectionné du nombre de pixels spécifiés dans le centre des flèches. Pour changer du nombre de pixels, cliquez sur le numéro.

Relayer Les objets de la sélection seront réarrangés, du premier au dernier ou du dernier au premier, dans l'ordre où ils ont été sélectionnés. Utilisez ces boutons pour définir l'ordre de tabulation d'un ensemble d'objets. Pour plus d'informations sur l'ordre de tabulation, consultez la section sur le focus du clavier ci-dessous.

4.1.3 Le Focus Clavier (The Keyboard Focus)

Le focus est une indication à l'utilisateur affichant quel contrôle recevra la frappe clavier. Précisément, quels sont les objets susceptibles de recevoir le focus clavier dépend du système d'exploitation actuel, et les propriétés appliquées au contrôle. L'édition des champs peut recevoir le focus, tout comme tous les autres objets sur Windows et Linux, et de nombreux objets sur Mac OS.

L'ordre dans lequel l'utilisateur se déplace grâce à des contrôles qui peuvent recevoir le focus clavier est déterminé par le calque de l'objet. Lorsqu'une carte est ouverte LiveCode pointe automatiquement le premier objet sur la carte capable de recevoir le focus clavier.
Vous pouvez activer la capacité d'un objet pour obtenir le focus du clavier en activant l'option focus avec le clavier dans l'inspecteur de l'objet, ou en définissant sa propriété de **traversalOn** via script.

Sur certaines plateformes les objets donnent une indication qu'ils ont le focus en affichant une bordure externe. Vous pouvez spécifier si une bordure doit être affichée en réglant l'option **Show Focus Border** dans l'Inspecteur, ou en définissant sa propriété **showFocusBorder** via script.

4.2 Objets et Types de Contrôles (Object & Control Types)

4.2.1 Piles - des fenêtres d'affichage, des palettes et des boîtes de dialogue

Dans LiveCode, chaque fenêtre est une pile. Cela inclut les fenêtres éditables, les boîtes de dialogue non modales et modales et les palettes, ainsi que des sous-fenêtres disponibles sur certains systèmes d'exploitation, tels que les panneaux et les tiroirs.

Astuce : *Si vous voulez placer des contrôles à l'échelle de la pile et que leurs positions respectives se modifient automatiquement lorsque vous redimensionnez la pile, consultez la section sur le gestionnaire de géométrie*

Cette rubrique traite de la façon dont les fenêtres sont mises en œuvre dans les applications de LiveCode ; comment modifier l'apparence de la fenêtre, et la façon de contrôler le comportement des différents types de fenêtres. Ce sujet ne couvre pas l'organisation des piles dans un fichier de pile, qui est couvert en détail dans la section Structurer votre application.

Attention : Ne commencez pas votre nom de pile avec **rev**. Les noms commençant par « rev » sont réservés par l'environnement de développement LiveCode.

Vous créez une nouvelle pile - qui peut alors être affichée dans n'importe quel type de modes (comme décrit ci-dessous) - en choisissant **File > New Mainstack**. Vous pouvez modifier les propriétés de la pile en choisissant **Object > Stack Inspector**.

4.2.2 Types de Fenêtres et Mode d'une Pile

La nature de la fenêtre de la pile dépend de la propriété de style de la pile et quelle commande a été utilisée pour ouvrir la pile. Vous pouvez spécifier le type de fenêtre lors de l'ouverture de la pile, ou passer la propriété de style de la pile afin de déterminer le type de fenêtre devant être affichée.

Note : Nous vous recommandons de spécifier le mode que vous souhaitez pour l'ouverture de votre pile dans la commande que vous utilisez pour l'ouvrir plutôt que d'utiliser la propriété de style. Cela rend plus facile de passer une pile du mode édition, dans l'environnement de développement, vers, par exemple, une boîte de dialogue dans l'environnement d'exécution.

4.2.3 Les Types de fenêtre standard

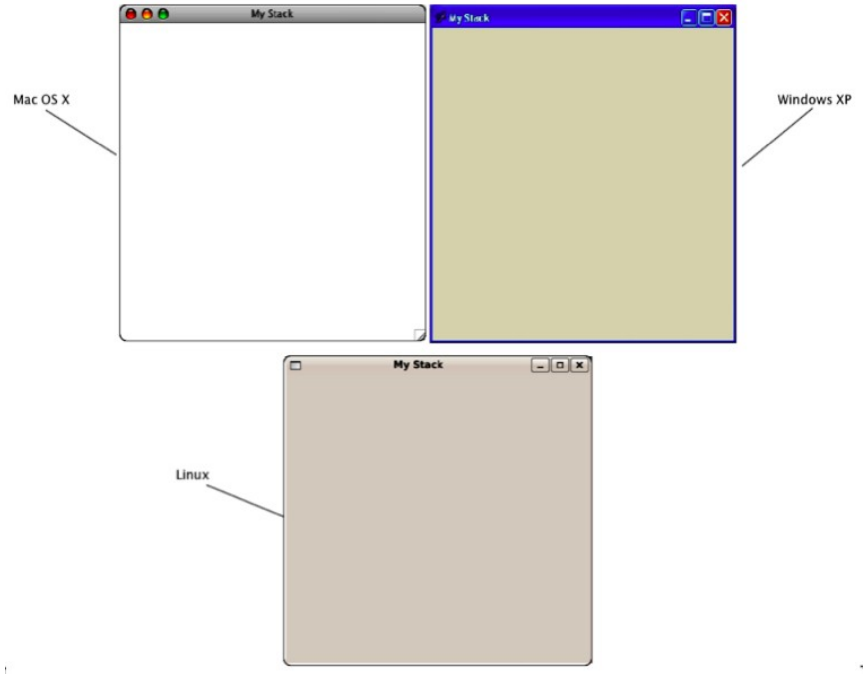
Les fenêtres LiveCode sont généralement de quatre types : fenêtres éditables ou **topLevel**, les boîtes de dialogue **modales** ou **non modales** ou fenêtres **palette**.

Important : Vous allez normalement créer une nouvelle pile et l'éditer alors qu'elle est en mode édition. Si vous souhaitez créer une boîte de dialogue, créez la pile comme n'importe quel autre pile. Ensuite, lors de son ouverture, spécifiez dans le script qu'elle devrait être affichée sous forme de boîte de dialogue, ou d'une palette, etc. Les commandes appropriées pour ce faire sont détaillées pour chaque type de fenêtre ci-dessous. Vous pouvez tester ces commandes lorsque vous travaillez sur votre mise en forme de la fenêtre et son script, en utilisant la boîte de message (voir la section du même nom), ou en utilisant le menu contextuel de la fenêtre (voir les boîtes de dialogue modales, ci-dessous) Pour plus de détails sur l'écriture de scripts en général, voir la section sur le codage de LiveCode

La plupart des fenêtres sont éditables ou **topLevel**, ce qui est le mode par défaut des piles de LiveCode. Si vous ouvrez une pile en utilisant la commande **go** (sans spécifier un mode), ou en utilisant l'option de menu **Open Stack**, alors la pile est affichée comme une fenêtre éditable à moins que sa propriété de style spécifie un autre type de fenêtre

4.2.4 Fenêtres Éditables – pour les documents

Une fenêtre modifiable a l'apparence et le comportement d'une fenêtre document standard. Elle peut être intercalée avec d'autres fenêtres, et vous pouvez utiliser l'un des outils de LiveCode pour créer, sélectionner, déplacer ou supprimer des objets dans la pile.



Fenêtres d'information éditables sur de multiples plateformes

Pour afficher une pile comme fenêtre éditable, utiliser les commandes **topLevel** ou **go** :

topLevel stack "My Stack"
go stack "My Stack" **-- "topLevel" est le mode par défaut**
go stack "My Stack" as **topLevel**

Les piles dont le style la propriété est définie sur "**topLevel**" toujours s'ouvrira comme une fenêtre modifiable, quelle que soit la commande que vous utilisez pour l'ouvrir.

Remarque : Si la propriété **cantModify** de la pile est définie sur vrai, la fenêtre conserve son aspect standard, mais d'autres outils que l'outil **Browse** ne peuvent plus être utilisés en elle. En d'autres termes, chaque outil se comporte comme l'outil **Browse** en cliquant dans la fenêtre d'une pile non modifiable.

4.2.5 Boîtes de Dialogue non modales – pour les alertes et les réglages

Les boîtes de dialogue non modales sont semblables aux fenêtres modifiables. Comme ces dernières, elles peuvent être entrelacées avec d'autres fenêtres de l'application. Leur apparence peut légèrement différer de l'apparence des fenêtres modifiables, en fonction de la plate-forme.

Comme les piles non modifiables, les boîtes de dialogue non modales ne permettent pas l'utilisation d'autres outils que l'outil ***Browse*** (parcourir). Utilisez les boîtes de dialogue non modales dans votre application pour des fenêtres comme **Preferences** ou une boîte de dialogue de recherche, qui ne nécessitent pas que l'utilisateur les révoque avant de faire une autre tâche.

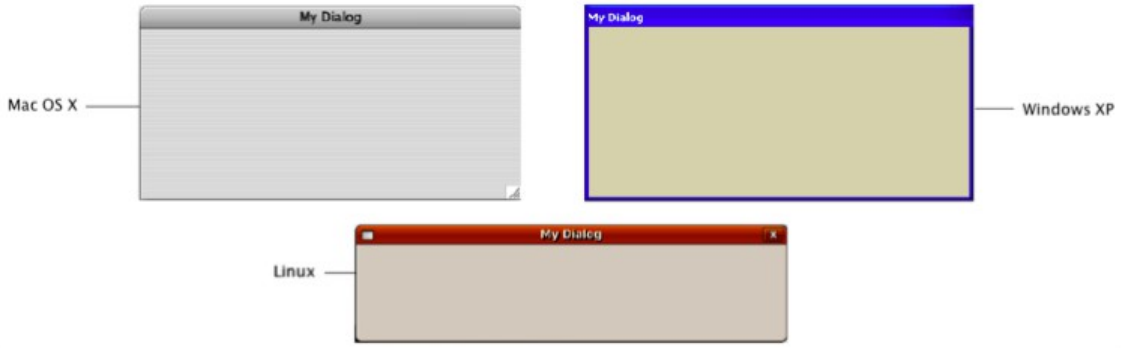
Pour afficher une pile dans une boîte de dialogue non modale, vous utilisez les commandes **modeless** ou **go** :

```
modeless stack "My Stack"
go stack "My Stack" as modeless
```

Les piles dont la propriété de style est réglée sur **modeless** s'ouvrent toujours comme des boîtes de dialogue non modales, quelle que soit la commande que vous utilisez pour les ouvrir.

4.2.6 Boîtes de dialogue Modales – pour alertes et réglages

Une boîte de dialogue modale est une fenêtre qui bloque d'autres actions pendant qu'elle s'affiche. Vous ne pouvez pas ramener une autre fenêtre de l'application au premier plan jusqu'à ce que la boîte de dialogue soit fermée, ni modifier la pile en utilisant les outils de la palette des outils. Alors qu'une boîte de dialogue modale est affichée, la commande à l'origine de son lancement, s'interrompt jusqu'à ce que la boîte de dialogue soit fermée.



Les Boîtes de Dialogue sur les Plateformes Multiples

Les boîtes de dialogue modales n'ont pas de commande de fermeture. Habituellement, elles contiennent un ou plusieurs boutons qui ferment la fenêtre quand on clique dessus.

Important : Si vous ouvrez par erreur une boîte de dialogue modale sans avoir inclus un bouton pour fermer la fenêtre, utilisez le raccourci du menu contextuel (**Ctrl + Maj + clic droit** pour Unix ou Windows, **Commande-Control-Maj-clic** sur Mac OS) pour afficher un menu contextuel. Choisissez **toplevel** pour rendre la pile modifiable.

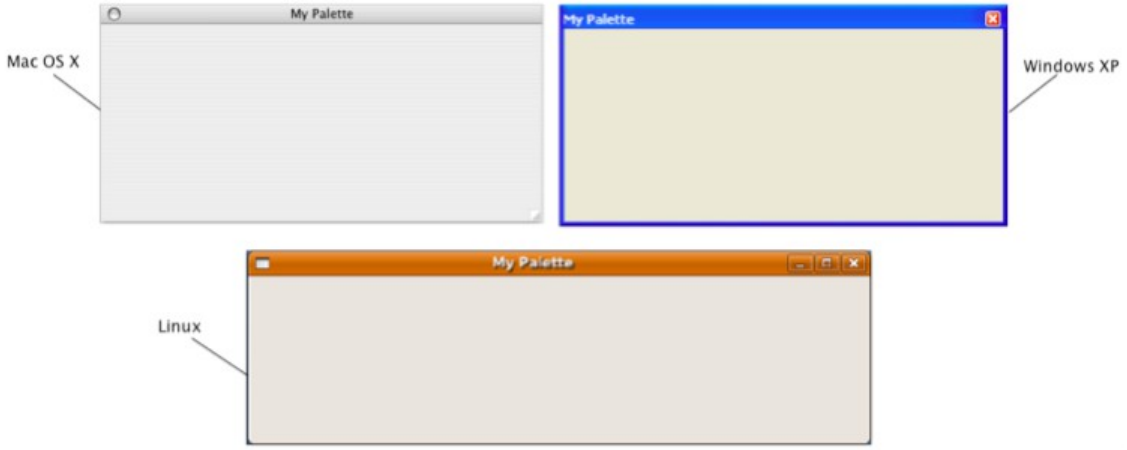
Pour afficher une pile dans une boîte de dialogue modale, utilisez la commande **modal** ou **go** :

```
modal stack "My Stack"
go stack "My Stack" as modal
```

Les piles dont la propriété de style est définie sur **modal** s'ouvriront toujours comme des boîtes de dialogue modales, quelle que soit la commande que vous utilisez pour les ouvrir.

4.2.7 Palettes – pour les accessoires et les fenêtres outils

Une palette a un aspect légèrement différent, avec une barre de titre plus étroite qu'une fenêtre éditable. Comme les fenêtres de zone de dialogue, une palette ne permet pas l'utilisation des outils autres que l'outil ***Browse*** (parcourir).



Palette Windows on Multiple Platforms

Une palette flotte devant les fenêtres modifiables ou les boîtes de dialogue non modales qui sont dans la même application. Même lorsque vous ramenez une autre fenêtre au premier plan, elle ne couvrira pas les palettes.

Note : Sur certains systèmes Unix, les palettes ont la même apparence et le comportement que les fenêtres ordinaires et ne flottent pas. Sur les systèmes Mac OS, les fenêtres palettes disparaissent lorsque leur application passe en arrière-plan et qu'une autre application passe au premier plan.

Pour afficher une pile dans une palette, utilisez la commande **palette** ou les commandes **go** :

palette stack "My Stack"
go stack "My Stack" as palette

Les piles dont la propriété de style est réglé sur **palette** s'ouvrent toujours comme palettes, indépendamment de la commande utilisée pour les ouvrir.

4.2.8 Boîte de Dialogue Question (Ask Question Dialog Boxes)

La boîte de dialogue poser une question est un type spécial de fenêtre qui est conçue pour faciliter toute demande de question à l'utilisateur. Il comprend une syntaxe spéciale pour l'ouverture de la boîte de dialogue avec le texte de la question et en retournant une réponse au script qui l'a appelé. Vous pouvez également spécifier le titre de la fenêtre, ainsi qu'une icône à afficher dans la fenêtre. La police, la position des objets, l'ordre des boutons et l'icône changera automatiquement en fonction du système d'exploitation. Toutefois, si vous n'avez pas besoin de plus de souplesse que prévu dans cette boîte de dialogue, vous devriez créer votre propre boîte de dialogue modal à la place (voir ci-dessus).

Boîte de dialogue question sur de multiples plateformes

Pour afficher la boîte de dialogue question, utiliser les commandes suivantes :

ask "What is your name?"
ask question "What is the answer?" titled "Question"

Le type de question détermine l'icône qui illustrera soit une question, soit une information soit une alerte d'erreur.

Pour changer la miniature montrée dans l'exemple Mac OS X ci-dessus, voir l'entrée **gREVAppIcon** dans le dictionnaire de LiveCode.

Le résultat est retourné dans une variable spéciale **it**.

if it is "Joe" **then** *doSomeThing*

Pour des détails complets sur la syntaxe, reportez-vous sur la commande **ask** dans le Dictionnaire du LiveCode.

Icônes pour les dialogues questions et réponses dans de multiples plateformes.

4.2.9 Dialogue Réponse / Alerte

Comme la boîte de dialogue **question** ci-dessus, la boîte de dialogue **réponse** est un dialogue spécial qui a été conçu pour le rendre facile l'affichage des informations dans une boîte de dialogue à l'écran et éventuellement permettre à l'utilisateur de faire un choix parmi une liste de sept choix maximum. La commande **answer** ouvre la boîte de dialogue, vous permet de spécifier le texte et les choix de bouton, et renvoie le clic sur le bouton utilisé pour le script qui l'a appelé. Vous pouvez également spécifier le titre de la fenêtre, ainsi que l' icône à afficher dans la fenêtre. Comme avec le dialogue question, la police, la position des objets, ordre des boutons et l'icône changeront automatiquement pour refléter le système d'exploitation. Toutefois, si vous n'avez pas besoin de la souplesse de cette boîte de dialogue, vous devriez créer votre propre boîte de dialogue modale à la place (voir ci-dessus).

Boîtes de dialogue Réponse sur Diverses Platerformes

```
answer "Salut tous le monde !"
answer question "Capitale de l'Italie ?" with "Paris" or "Londres" or "Rome" titled "Choix Multiple"
```

Le résultat est retourné dans la variable spéciale **it**.

```
if it is "Rome" then answer information "C'est la bonne réponse."
```

Pour des détails complets sur la syntaxe, voir la commande **answer** dans le dictionnaire de LiveCode.

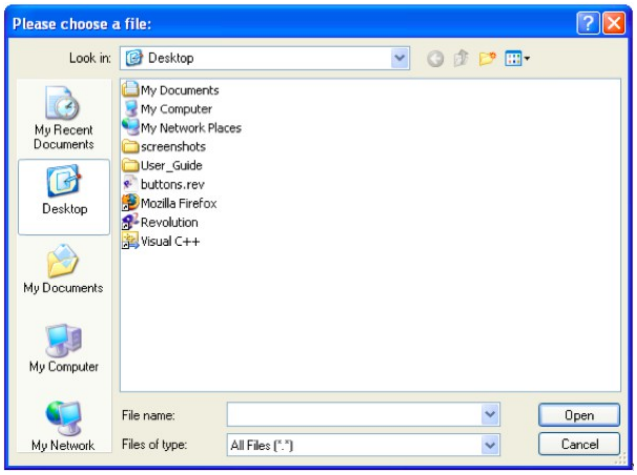
La boîte de dialogue réponse est mise en œuvre en interne comme une pile attaché à l'IDE LiveCode. Pour plus d'informations sur la personnalisation de l'IDE, voir la section sur Modification de l'interface utilisateur de LiveCode.

Astuce : Si vous n'êtes pas sûr du nom d'une pile, vous pouvez utiliser la fonction **mouseStack** pour le découvrir. Entrez les informations suivantes dans la boîte de message (**Tools > Message Box**) : **put the mouseStack** puis déplacez le pointeur de la souris sur la fenêtre de la pile et appuyez sur **Entrée**.

4.2.10 Sélecteur de Fichiers

Les boîtes de dialogue de sélection de fichier vous permettent d'afficher les boîtes de dialogue standard du système. Ces dialogues permettent à l'utilisateur de sélectionner un fichier ou un ensemble de fichiers, de sélectionner un répertoire, ou spécifier un nom et un emplacement pour enregistrer un fichier.

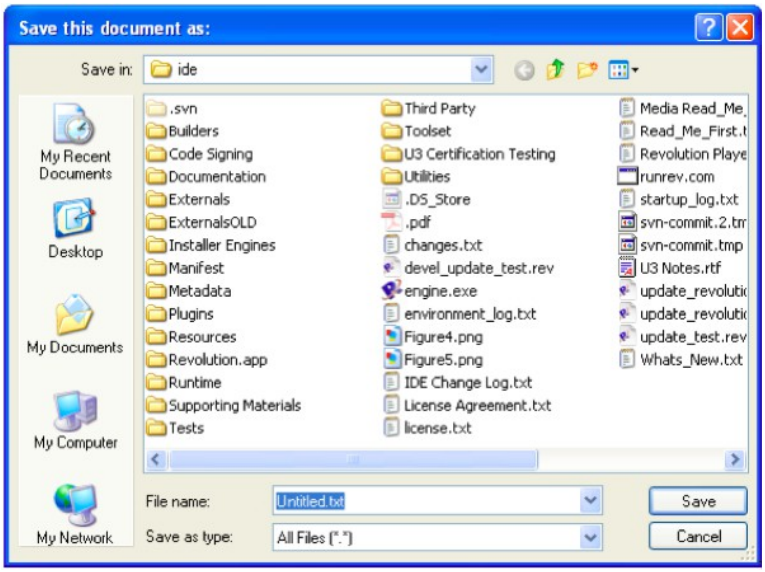
La syntaxe pour appeler les boîtes de dialogue de sélection de fichiers reprend la syntaxe des boîtes de question et de dialogue détaillées ci-dessus. Toutefois, contrairement à ces dialogues, les boîtes de dialogue de sélection de fichiers sont affichées en utilisant les boîtes de dialogue standard du système le cas échéant. L'exception notable au moment de la rédaction de ce document est la plateforme Linux, où un dialogue intégré est utilisé à la place en raison d'un support OS plus limité. Vous pouvez forcer les autres plates-formes à utiliser ce sélecteur de fichier basé sur la pile intégrée en définissant la propriété globale **systemFileSelector**.



Boîtes de Dialogue Sélection de Fichiers

```
answer file "Select a file:"
answer file "Select an image file:" with type "QuickTime Movies|mov" or type "All Files|"
```

Le chemin d'accès au fichier sélectionné par l'utilisateur est renvoyé dans la variable spéciale it. Si l'utilisateur a annulé la boîte de dialogue, la variable spéciale **it** sera vide et annuler sera retourné par la fonction de résultat.
Pour des détails complets sur la syntaxe, consultez le sélection de fichier en fonction du type de fichier (**answer with type**) dans le dictionnaire de LiveCode.



Sélecteur de Fichier pour sauvegarde de Fichier

ask file "Save this document as:" with "Untitled.txt"
answer file "Select an image file:" with **type** "Text Files|txt" or **type** "All Files|"

Le chemin d'accès au fichier à sauvegarder est retourné dans la variable spéciale **it**. Si l'utilisateur a annulé la boîte de dialogue, la variable **it** sera vide et annuler sera retourné par la fonction de résultat. Pour des détails complets sur la syntaxe, consultez le fichier question en fonction du type dans le dictionnaire de LiveCode.

answer folder "Please choose a folder:"
answer folder "Please choose a folder:"with "/root/default folder"

Le chemin du fichier dans le dossier sélectionné par l'utilisateur est renvoyé dans la variable spéciale **it**. Si l'utilisateur a annulé la boîte de dialogue, la variable sera vide et de l'information annuler sera retournée par la fonction de résultat. Pour des détails complets sur la syntaxe, reportez-vous répondre dossier dans le dictionnaire de LiveCode.

4.2.11 Boîte de Dialogue Sélecteur de couleur

Elle vous permet d'afficher boîte de dialogue de sélection de couleur standard du système d'exploitation.

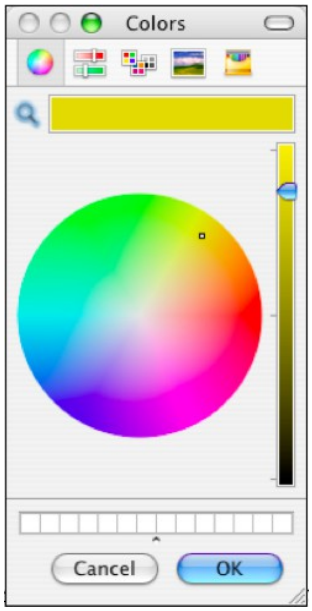
Dialogue Sélecteur de Couleur

```
answer color
```

La couleur choisie est retournée dans la variable spéciale **it**.

Si l'utilisateur a annulé la boîte de dialogue, it sera vide et annuler sera retourné par la fonction de résultat.

Pour des détails complets sur la syntaxe, voir **answer color** dans le dictionnaire de LiveCode.

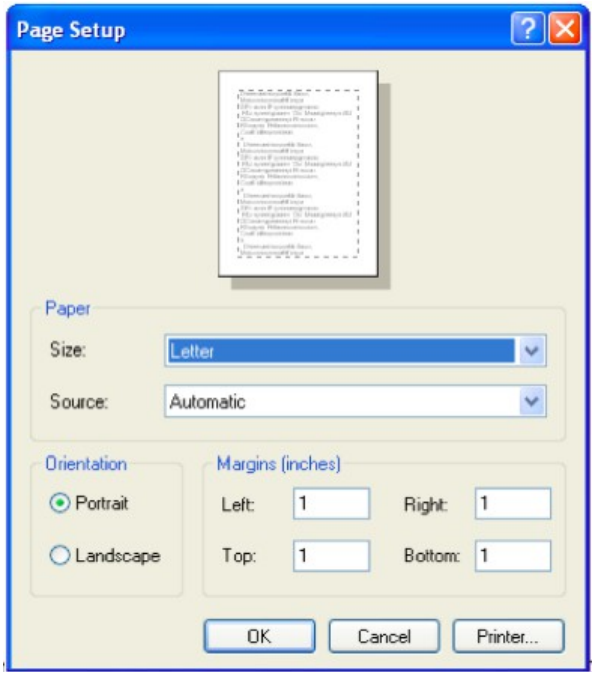


4.2.12 Boîte de Dialogue Imprimante

Les boîtes de dialogue de l'imprimante vous permettent d'afficher les boîtes de dialogue de configuration de l'imprimante et la page standard.

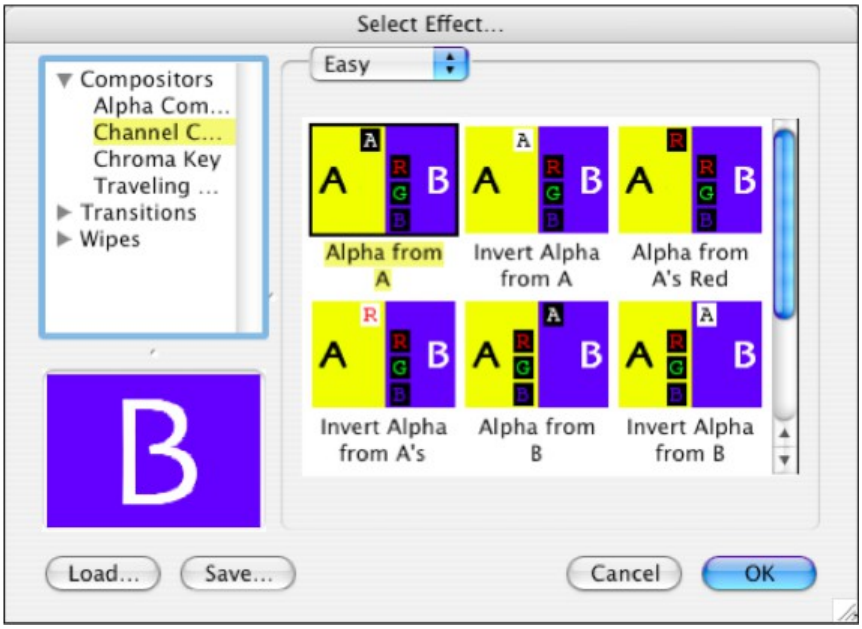
Answer printer pour ouvrir les boîtes de dialogue imprimantes standard

Utilisez la commande **answer printer** pour afficher une boîte de dialogue standard de l'imprimante avant l'impression. Si l'utilisateur annule les dialogue, annuler sera retourné par la fonction de résultat.



4.2.13 Visual Effect Dialog

Cette boite vous permet d'afficher la boîte de dialogue QuickTime effets spéciaux.



Boite de dialogue pour choisir un effet QuickTime

Utilisez la commande **answer effect** pour afficher une boîte de dialogue d'effet standard QuickTime.
L'effet sera retourné sous forme de données binaires dans le variable spéciale **it**.

Vous pouvez utiliser cette variable avec la commande **visual effect** pour créer une transition de l'effet visuel.

Si l'utilisateur annule les dialogue, **it** sera vide et annuler sera retourné par la fonction de résultat.

4.2.14 Fenêtre mélange Alpha (Infobulles et Multimédia Améliorés)

Utilisez l'option de forme dans l'inspecteur de la pile pour définir la propriété **windowShape** d'une pile sur le canal transparent ou alpha d'une image qui a été importée avec son canal alpha (par exemple au format PNG ou GIF). Cela vous permet de créer une fenêtre avec des «trous» ou une fenêtre avec une translucidité variable.

Important : La bordure et la barre de titre de la pile ne sont pas représentées, si la propriété **windowShape** de la pile est définie. Cela signifie que vous aurez besoin de fournir des procédés de glissement et de fermeture de la fenêtre si vous voulez que l'utilisateur soit en mesure de faire ces tâches.

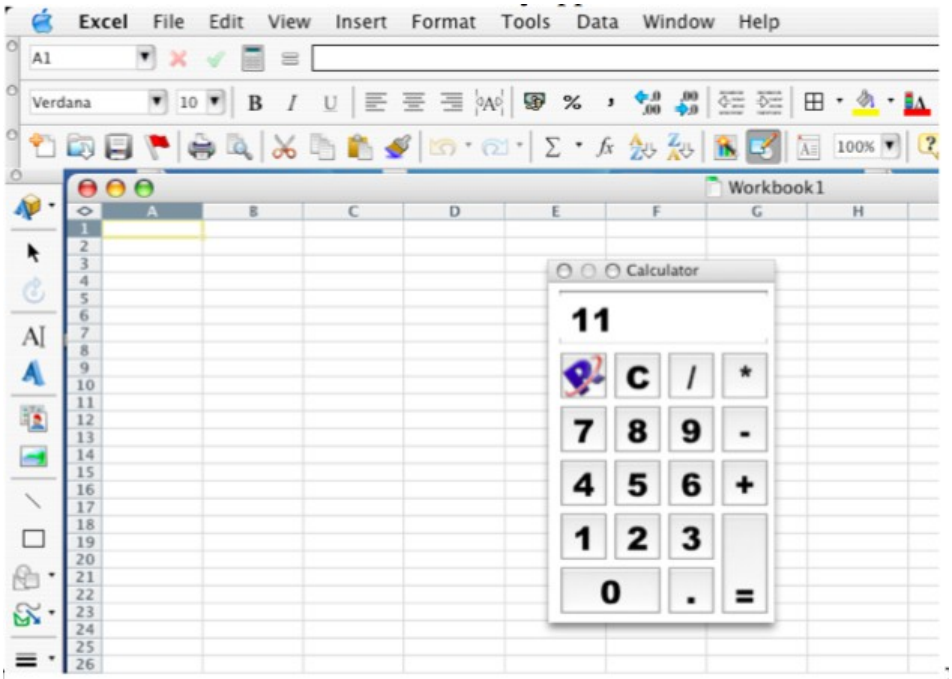
Fenêtre avec un masque alpha appliqué

Vous pouvez modifier la propriété **windowShape** dynamiquement par un script pour une série d'images pour créer une fenêtre translucide animé.

4.2.15 Palettes Systèmes (Palettes Flottantes)

Une palette système est comme une palette, à l'exception qu'elle flotte au premier plan de toutes les fenêtres à l'écran, et pas seulement de celles de sa propre application.

Vous utiliserez les palettes système pour les utilitaires que l'utilisateur devra être en mesure de voir et d'accéder depuis n'importe quelle application.



Palette système flottant au-dessus d'autres applications

Pour afficher une pile dans une palette système, vous activez la case à cocher de l'inspecteur "**Float Above Everything**" de la pile. Pour plus de détails sur cette fonctionnalité, consultez l'entrée **systemWindow** dans le dictionnaire de LiveCode.
L'utilisation de cette fonction l'emporte sur le style ou le mode de la pile. Le style de la palette système n'est actuellement pas pris en charge sur Linux et Unix.

4.2.16 Boîtes de dialogue de Type Fiche – Mac OS X seulement

Une fiche est comme une boîte de dialogue modale, si ce n'est qu'elle est associée à une fenêtre unique, plutôt qu'à l'ensemble de l'application. Une fiche apparaît dans sa fenêtre parente, glissée par-dessous la barre de titre. Lorsque la fiche est visible, elle bloque les autres actions dans sa fenêtre parente, mais pas dans d'autres fenêtres de l'application. Pour afficher une pile dans une boîte de dialogue de type fiche, vous utilisez la commande **sheet** :

```
sheet "My Stack" -- appears in defaultStack
sheet "My Stack" in stack "My Document"
```

Remarque: Notez que les commandes **answer**, **answerfile**, **answerfolder**, **ask**, **askfile**, et **answerfolder** (voir ci-dessus) sont toutes dotées d'une forme de types fiche **as sheet**, de sorte que vous pouvez afficher toutes ces boîtes de dialogue comme des fiches sur Mac OS X. Vous pouvez utiliser en toute sécurité la forme **as sheet** pour une application multi-plateforme autant que sur des systèmes autres que OS X, la commande **sheet** affiche la pile comme une boîte de dialogue modale ordinaire.

```
answer the long system date as sheet
```

La fiche se déroule sous la barre de menu de la fenêtre.

4.2.17 Tiroirs - Mac OS X seulement

Un tiroir est une fenêtre secondaire qui coulisse du dessous d'un bord de fenêtre, et revient en coulissant lorsque vous le fermez. Vous utilisez généralement un bouton dans la fenêtre principale pour ouvrir et fermer un tiroir.
Pour afficher une pile comme un tiroir, vous utilisez la commande **drawer** :

```
drawer "My Stack" at left -- of defaultStack
drawer "My Stack" at bottom of stack "My Document"
```

Sur les systèmes d'exploitation autres que OS X, la commande **drawer** : affiche la pile comme une fenêtre modifiable. Parce que cela ne correspond pas bien à d'autres plates-formes, nous vous recommandons de n'utiliser les tiroirs que pour les applications qui sont en cours de développement pour Mac OS X. Utilisez tiroirs pour contenir les paramètres, les listes de favoris, et des contrôles similaires qui sont accessibles fréquemment, mais qui n'ont pas besoin d'être constamment visible.

4.2.18 Menu pile - pour affichage des menus non standard

Il est également possible d'afficher une pile comme un menu déroulant, contextuel ou comme menu des options. Un menu pile est utilisé quand un menu doit contenir d'autres choses que juste du texte. Par exemple, un menu déroulant contenant un curseur ou un menu d'options comprenant des icônes au lieu du texte, peut être mis en œuvre sous la forme d'un menu pile. Pour afficher un menu pile, vous créez un bouton et définissez sa propriété **menuName** du nom de la pile. Lorsque l'utilisateur clique sur le bouton, la pile est affichée avec le comportement d'un menu. En interne, le menu est implémenté comme une fenêtre, et vous pouvez utiliser le menu contextuel, le menu déroulant, ou une commande d'option pour afficher une pile comme l'un de ces types de menu.

Note: Généralement un menu dans une application LiveCode est implémenté comme un bouton. Nous recommandons que les menus soient mis en œuvre à l'aide des boutons, car ceux-ci seront automatiquement dessinés avec la forme du thème natif de chaque plate-forme. Pour plus de détails, voir la section sur les menus ci-dessous

4.2.19 Agencement des Piles – Apparence des Fenêtres

Les ornements de pile vous permettent de spécifier la façon dont la barre de titre et la bordure d'une fenêtre seront établis. Vous pouvez accéder aux options **decorations** de la pile via **Object** > **Stack Inspector**. Outre les différences d'apparence entre les différents modes de fenêtre, l'apparence et les fonctionnalités d'une fenêtre peuvent varier en fonction des propriétés de la pile qui est affichée et de la plate-forme. La barre de titre d'une fenêtre affiche le titre de la fenêtre, ainsi qu'une case de fermeture, de minimisation, de maximisation et de réduction.

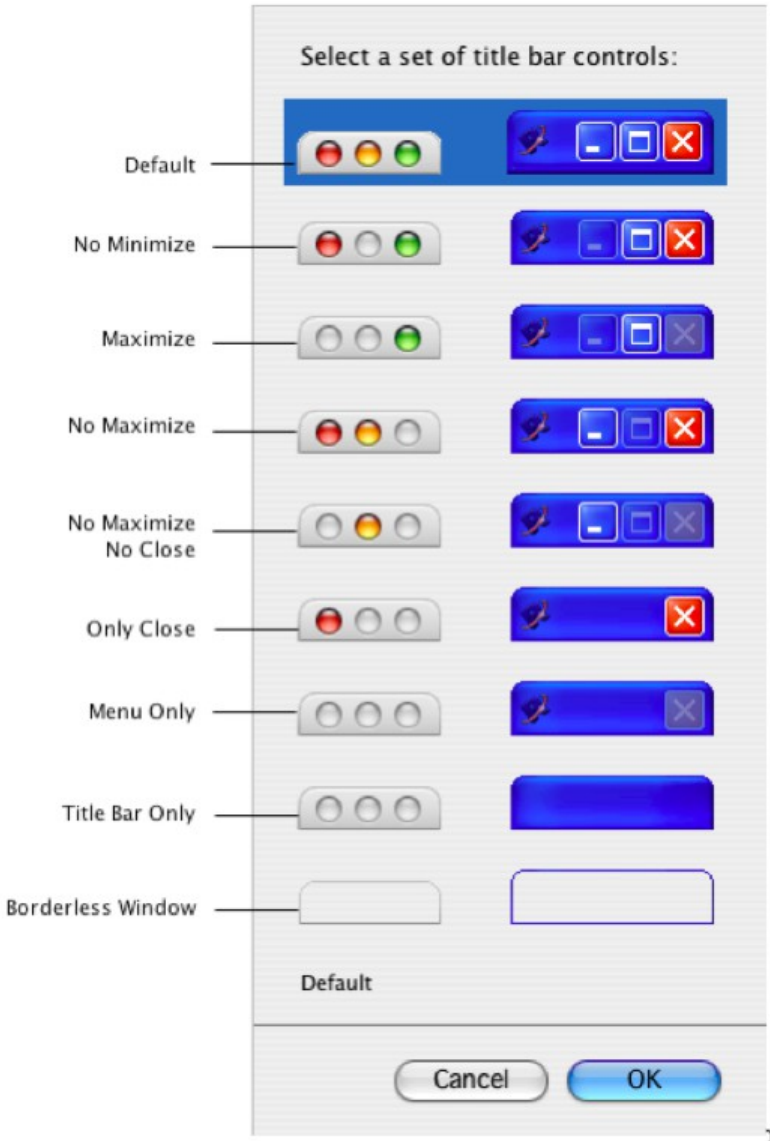
Les propriétés ci-dessus peuvent également être définies par script, pour plus de détails, voir la propriété

decorations dans le dictionnaire de LiveCode.

Vous pouvez spécifier une ou plusieurs apparence suivantes :

- **title** : Affiche la barre de titre de la fenêtre
- **minimize** : Affiche le bouton minimiser (iconifier) dans la barre de titre
- **maximize** : Affiche le bouton maximiser (zoom) la barre de titre
- **close** : Affiche le bouton de fermeture dans la barre de titre
- **menu** : Affiche la barre de menus de la fenêtre (Unix et Windows uniquement)
- **system** : Affiche la fenêtre comme une fenêtre système (OS OS X, Unix et Windows uniquement)
- **noShadow** : Retire l'ombre tombante de la fenêtre (OS OS X uniquement)
- **metal** : Affiche la fenêtre avec un aspect texturé (OS OS X uniquement)

```
set the decorations of this stack to "title,minimize"
```



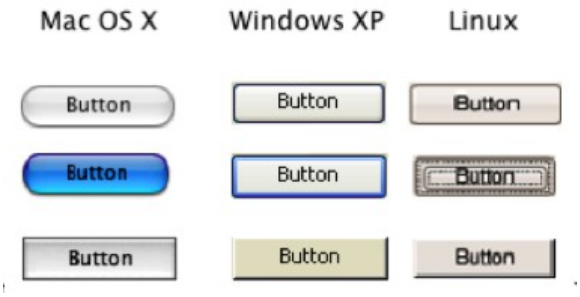
Apparence des Fenêtres sur Mac OS X et Windows

Bien que le mode de la pile soit distinct de ses ornements, le mode peut avoir une incidence si ces propriétés ont un effet. Si la propriété **decorations** est réglée sur "par défaut", elle affiche les l'apparence appropriée pour le type de fenêtre en cours sur la plateforme utilisée

Note: Sur Mac OSX, la propriété de l'ombre d'une pile permet de contrôler si la fenêtre de la pile a une ombre portée. Sur les systèmes OS X, vous pouvez définir cette propriété à false pour créer une fenêtre sans ombre

4.2.20 Contrôles bouton - pour effectuer des actions

Un bouton est un objet cliquable qui est généralement prévu pour permettre à un utilisateur d'effectuer une action en cliquant



Objets Boutons sur les différentes plateformes

Les cases à cocher et les boutons radio sont utilisés pour permettre à l'utilisateur de faire des choix. Les boutons radio sont utilisés lorsqu'une seule option sur un ensemble peut être sélectionnée à tout moment. Les cases à cocher sont utilisées lorsque certaines options peuvent être activées tandis que d'autres peuvent être éteintes.

Note : *LiveCode applique automatiquement la règle de mise en évidence d'un seul bouton radio à la fois si vous placez les boutons radio dans un groupe. Pour plus de détails sur les groupes, voir la section sur les groupes et les arrière-plans.*

Cases à cocher et boutons Radio sur Diverses Plateformes

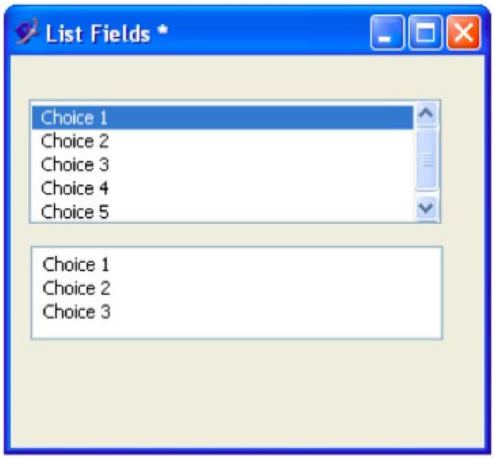
Tous les objets boutons dans LiveCode sont très flexibles et personnalisables. Les paramètres communs incluent la possibilité d'afficher et de masquer la bordure ou le remplissage, et d'afficher une icône.
Les icônes vous permettent de fournir un large éventail de fonctionnalités. Par exemple, vous pouvez créer un effet ***roll over*** en réglant la propriété **hovericon**. Ou vous pouvez créer une case à cocher spécifique en positionnant une icône plus une seconde icône en surbrillance - cela permet de remplacer le bouton du système standard en affichant votre icône dans chaque état selon que le bouton ait été pressé ou relâché.

Important: Les icônes des boutons ne sont pas limitées en largeur ou en hauteur. Ils peuvent être animés en utilisant une animation GIF. En fait, une icône peut référencer une image contenue dans le fichier de la pile LiveCode. Référencer une image de cette façon permet d'économiser l'espace disque et vous permet de mettre à jour toutes les icônes dans votre pile en mettant à jour une seule image. Voir le chapitre Travailler avec les médias pour plus d'informations.

4.2.21 Contrôles du champ texte - pour l'affichage ou la saisie de texte

Les champs vous permettent d'afficher du texte. Les champs peuvent éventuellement permettre à l'utilisateur de modifier le texte.
Les champs gèrent plusieurs polices, styles et couleurs, des images et un sous-ensemble de balises **HTML** de base.
Les champs peuvent être reliés à une base de données en y accédant directement avec la bibliothèque de base de données.
Ils peuvent afficher et avoir un rendu **XML** en utilisant la bibliothèque **XML**.
Les champs de liste permettent à l'utilisateur de sélectionner un ou plusieurs choix.
Les champs tableaux permettent l'affichage des données similaires à une feuille de calcul.
D'autres types de champ peuvent facilement être créés, y compris les vues en arborescence, ou tout hybride entre ces différents types, avec l'aide d'un petit script.
Il y a aussi une bibliothèque disponible qui vous permet d'héberger un navigateur web au sein d'un objet et de le contrôler par script.

4.2.22 Contrôles des Champs de Liste et de table

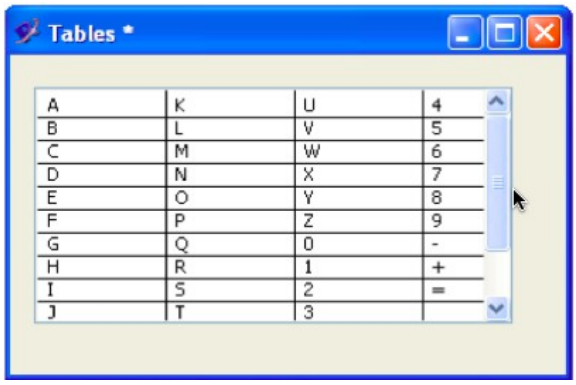


Champs de Liste

Les champs de liste permettent d'afficher plusieurs choix. Les utilisateurs ne peuvent pas modifier les champs de la liste.

Vous pouvez spécifier si l'utilisateur est autorisé à effectuer une seule sélection ou plusieurs sélections.

4.2.23 Contrôle de Champ detable – pour afficher unetable de données



Champ detable

Les champs detable vous permettent d'afficher des données dans les cellules, et éventuellement permettre à l'utilisateur de modifier les cellules.

Les champs detable sont idéaux pour afficher des données sous forme de tableaux basiques.

Pour afficher des données plus complexes, nous vous recommandons d'utiliser le contrôle de grille de données.

4.2.24 Contrôle Grilles de Données (Data Grid) - pour présenter des données

Les grilles de données permettent d'afficher des données à la fois dans la grille et dans des formulaires. Vous pouvez personnaliser une grille de données pour inclure des agencements personnalisés qui incluent tout autre objet LiveCode.
Les grilles de données peuvent fournir une vue depuis une base de données, vous permettant d'afficher des ensembles volumineux de données. Pour une documentation complète sur l'utilisation de grilles de données, consultez la documentation en ligne à l'adresse:

http://lessons.runrev.com/m/datagrid

Data Grid

4.2.25 Cartes

Chaque pile contient un ou plusieurs panneaux de commandes ou écrans distincts. Chaque écran est connu comme une carte. Chaque carte peut avoir un aspect tout à fait différent, ou toutes les cartes dans une pile peuvent partager tout ou partie des éléments, en utilisant des groupes partagés, appelés environnement (voir ci-dessous).
Choisir **Object** > **New Card** pour créer une nouvelle carte dans la pile actuelle. La nouvelle carte sera soit

complètement vide soit contiendra des groupes partagés de la carte précédente.

4.2.26 Groupes & Arrière-plan - pour l'organisation et les contrôles de partage

Les groupes, le type d'objet le plus polyvalent de LiveCode, sont utilisés à plusieurs fins : des grappes de boutons radio, des barres de menus, pour la création de zones d'objets défilants dans les cartes, et comme fond d'écran pour afficher des ensembles d'objets qui sont partagés entre les cartes. Les groupes peuvent également être utilisés pour créer une carte simple et une base de données de la pile, en organisant les champs qui contiennent un dossier différent sur chaque carte.

Qu'est-ce qu'un groupe?

Un groupe est un objet simple qui contient un ensemble d'objets. Les objets sont regroupés en sélectionnant les commandes que vous souhaitez inclure dans le groupe, puis en utilisant la commande de groupe ou en choisissant **Object** / **Group Selected**.

Une fois que vous avez créé le groupe, il devient un objet à part entière. Vous pouvez sélectionner, copier, déplacer et redimensionner le groupe, et tous les objets du groupe venu avec elle. Les objets du groupe maintiennent leurs propres identités, et vous pouvez ajouter des objets au groupe ou les supprimer, mais les objets sont détenus par le groupe à la place de la carte.

Un groupe possède ses propres propriétés et son propre script. Les groupes peuvent être de toute taille, peuvent être affichés ou masqués, et peuvent être déplacés n'importe où dans la fenêtre de la pile, comme n'importe quel autre contrôle.
Comme d'autres contrôles, les groupes peuvent être superposés dans n'importe quel ordre avec les autres contrôles sur la carte. Les groupes peuvent également afficher une bordure autour d'un ensemble d'objets.

Groupe de bouton radio avec un titre et une bordure de groupe

Contrairement à d'autres contrôles, cependant, les groupes peuvent apparaître sur plus d'une carte.

Vous placez un groupe sur une carte en utilisant la commande **place** ou le sous-menu **Place Group** dans le menu **Object**.

Important : Un groupe qui est partagé entre les cartes apparaît au même emplacement sur chaque carte. Une modification apportée à la position d'un groupe partagé sur une carte se reflète sur toutes les autres cartes qui partagent le groupe.

Les groupes et fonds d'arrière-plans

Tant le terme de groupe (**group**), que celui d'arrière-plan ou fond (**background**) peut être utilisé pour désigner des groupes. Ces termes sont interchangeables dans certaines circonstances et signifient des choses différentes dans d'autres. Les différences sont expliquées plus en détail ci-dessous.

En général, le terme groupe se réfère à des groupes qui sont placés sur une carte, tandis que le terme de fond se réfère à tous les groupes dans une pile qui sont disponibles pour une utilisation en arrière-plan (voir ci-dessous).
L'expression **the number of groups** évaluent le nombre de groupes dans la carte en cours. L'expression **the number of backgrounds** évalue le nombre de groupes en arrière-plan dans la pile actuelle, y compris les groupes qui ne sont pas placés sur la carte actuelle.

Astuce : Lorsque vous faites référence à un groupe **by number**, si vous utilisez le mot **group**, le nombre est interprété comme désignant les groupes sur la carte référencée, ordonnés par couche (**layer**). Si vous utilisez le mot **background**, le nombre est interprété comme désignant les groupes de la pile, dans l'ordre de leur création**.**

Astuce : Par exemple, l'expression **the name of group 1** évalue le nom du groupe du plus bas-niveaux sur la carte actuelle, alors que l'expression **the name of background 1** renvoie le nom du premier groupe qui a été créé dans la pile - que ce groupe particulier soit placé sur la carte en cours, ou bien sur n'importe quelle carte de l'ensemble**.**

Le terme de **background** peut être également utilisé pour désigner l'ensemble des cartes qui partagent un groupe particulier. La déclaration suivante va à la troisième carte sur laquelle le groupe nommé "Navigation" est placé :

```
go card 3 of background "Navigation"
```

Les groupes imbriqués

LiveCode supporte les groupes imbriqués (un groupe contenant un autre). Un groupe étant déjà lui-même un contrôle qui peut être inclus dans un autre groupe. La création d'un groupe imbriqué est identique à la création d'un groupe simple : sélectionnez les contrôles que vous souhaitez regrouper (y compris le groupe déjà existant), puis choisissez Object > Group Selected. Le groupe actuel est maintenant un membre du nouveau groupe.

Sélection et modification de groupes

Pour sélectionner un groupe, cliquez simplement sur l'un des objets qui sont contenus en son sein. Cela permet de sélectionner le groupe.
Si vous souhaitez sélectionner un objet dans le groupe, au lieu du groupe lui-même, il ya deux façons de le faire :

- Vous pouvez activer l'option **Select Grouped** dans la barre d'outils ou dans **Edit** / **Select Grouped Controls** . Cela entraîne que les groupes seront ignorés lors de la sélection des objets, ce qui vous permettra de sélectionner des objets à l'intérieur du groupe, comme si le groupe n'existait pas. Vous pouvez accéder à ce mode par un script en activant la propriété globale **selectGroupedControls**.
- Sinon, vous pouvez passer en mode **edit group**, un mode spécial qui n'affiche que les objets à l'intérieur de ce groupe. Sélectionnez le groupe, puis appuyez sur **Edit Group** sur la barre d'outils ou choisissez **Object** / **Edit Group**. Lorsque vous avez terminé, choisissez **Object** / **Arrêter groupe d'édition**. Vous pouvez activer ou désactiver ce mode par programmation en utilisant **editing commandsstart** et **stop editing**.

Astuce : Si la bordure du groupe a été activée, un contour trace les bords du groupe. Cependant, cliquer à l'intérieur ou sur la bordure ne sélectionne pas le groupe. Pour sélectionner le groupe, vous devez cliquer sur un de ses contrôles.

Mise en place et retrait des fonds d'arrière-plan

Une fois que vous créez un groupe, vous pouvez l'afficher sur tout ou partie des cartes dans la pile. Tout d'abord, assurez-vous que la propriété **Behave** du groupe ait l'option **Background** activée dans l'inspecteur. Puis accédez à la carte où vous souhaitez placer le groupe et choisissez **Object** / **Place Group** pour placer une instance d'un groupe particulier sur la carte actuelle. Vous pouvez contrôler ces fonctions à partir du script en utilisant la propriété **backgroundBehavior** et la commande **place**.

Note : Lorsque vous créez une nouvelle carte, s'il existe des groupes sur la carte actuelle avec les options **Behave** et **Background** activées, ceux-ci sont automatiquement placés sur la nouvelle carte. Pour faciliter le partage d'un même groupe pour toutes les cartes dans une pile, créez le groupe sur la première carte et définissez cette propriété sur vrai, avant de créer toutes les autres cartes.

Pour supprimer un groupe de la carte actuelle sans le supprimer de la pile sélectionnez le groupe et choisissez **Object** / **Supprimer le groupe**. Le groupe disparaît de la carte actuelle, mais il est toujours placé sur toutes les autres cartes qui partagent le groupe. Vous pouvez supprimer un groupe par script avec la commande **remove**.

Astuce : Vous pouvez utiliser la commande **start editing** de la boîte de message pour modifier un groupe non placé, sur aucune des cartes. Puisque le groupe ne figure sur aucune carte, vous devez y faire référence en utilisant le terme **background** au lieu du terme **group**.

Vous pouvez supprimer complètement un groupe de la même manière que vous supprimez tout autre objet, en sélectionnant le groupe et choisissez **Edit** / **Clear** ou bien en pressant la touche retour du clavier.

Important : La suppression d'un groupe **background** le supprime de toutes les cartes où il apparaît et de la pile elle-même.

Pour dissocier un groupe de nouveau, sélectionnez le groupe et choisissez **Object > Ungroup**. Vous pouvez dissocier un groupe par un script utilisant la commande **ungroup**.
La dissociation supprime l'objet du groupe et de ses propriétés (y compris son script) de la pile, mais ne supprime pas les contrôles en elle. Au lieu de cela, ils deviennent des commandes de cartes de la carte actuelle.
Les contrôles disparaissent de toutes les autres cartes du groupe.

Remarque : Si vous dissociez un groupe, puis sélectionnez les contrôles et les regroupez avant de quitter la carte actuelle, le groupe est restauré tel qu'il était. Toutefois, quitter la carte actuelle rend le dégroupage permanent et supprime le groupe de toutes les autres cartes concernées.

Groupes et le chemin du message

Pour plus de détails sur la façon dont les groupes et les arrières-plans s'inscrivent dans le chemin du message, voir la section sur les groupes, Arrière-plans et chemin du message ci-dessous.

4.2.27 Objets Graphismes, Images, Lecteurs, Audio & Vidéo clip

LiveCode prend en charge un large éventail de formats de médias, vous permettant de produire des applications multimédia. L'objet image vous permet d'importer ou de référencer des images, de les manipuler par script ou interactivement avec les outils de dessin, et de les enregistrer dans différents formats avec des options de compression variables. Le support s'étend à la couche alpha des images PNG et des images GIF animées. Les images peuvent être importées et réutilisées dans une pile pour créer des éléments d'interface personnalisés ou interactifs. Pour apprendre à travailler avec ces objets dans un script, consultez la section sur le travail avec les médias.

Les formats de fichiers image supportés sont GIF, JPEG, PNG, BMP, XWD, XBM, XPM, ou PBM, PGM, PPM . Sur les systèmes Mac OS, les fichiers PICT peuvent également être importés (mais ils ne peuvent pas être affichés sur les systèmes UNIX ou Windows). Pour plus de détails sur chacun de ces formats, consultez la section sur le travail avec les médias.

Vous pouvez importer des images en utilisant **File > Import as Control > Image File**.
Vous pouvez référencer une image en utilisant **File > New Referenced Control > Image fil**.

Les outils de dessin ne peuvent être utilisés que sur les images qui ont été importées comme contrôle. Voir la

section Utilisation des outils de dessin pour plus de détails sur la façon d'utiliser les outils de dessin.

Les graphiques vectoriels peuvent également être créés et manipulés avec les outils graphiques et par script. LiveCode prend en charge les chemins avec les variables **fills**, **gradients**, **blended** et, **antialiased** graphics. Au moment d'écrire ces lignes, il y a aussi une bibliothèque tierce partie qui permet l'importation et l'exportation des graphiques au format SVG. Utilisez des objets graphiques pour créer des interfaces interactives, des graphiques, des tableaux ou des jeux.

Utilisez l'objet **player** pour afficher et interagir avec tous les formats multimédias pris en charge par QuickTime. LiveCode vous permet d'activer et de désactiver les pistes dans un film, le panoramique, le zoom ou de changer l'emplacement dans une séquence QTVR, de définir des messages de rappel qui déclenchent des scripts à des points spécifiques dans le film, et de diffuser des films à partir d'un serveur. Au moment d'écrire ces lignes, il y a aussi une bibliothèque de tierce partie qui vous permet d'éditer et de sauvegarder des films par script.

Les objets **Audio Clip** & **Video Clip** vous permettent d'intégrer des données de clip audio ou vidéo dans une pile. Certains formats de clips audio peuvent être lus directement sans installation de QuickTime. Ils n'ont pas de représentation visuelle et sont accessibles par script ou dans le navigateur de l'application. Les lecteurs, les objets **Audio Clip** & **Video Clip** sont abordés dans leurs sections respectives dans le chapitre sur le travail avec les médias.

4.2.28 Contrôles du menu - pour afficher des choix

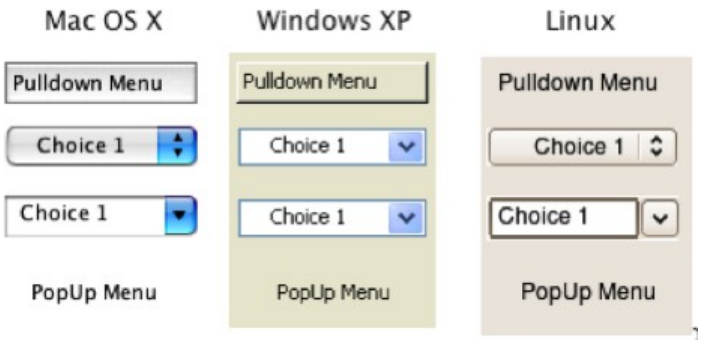
Les menus sont utilisés pour afficher une liste de choix. Le menu déroulant affiche un menu déroulant standard, et peut être inséré automatiquement dans la barre du menu principal sur les systèmes Mac OS. Le menu d'options permet de choisir parmi une liste. La liste déroulante permet à l'utilisateur de taper une option ou choisir parmi une liste. Les menus contextuels peuvent être affichés sous le curseur et utilisés pour fournir des options contextuelles n'importe où dans votre application. Pour plus d'informations sur l'utilisation des menus déroulants dans la barre de menu principal, voir la section sur le **Menu Builder** ci-dessous.

Le contenu des menus peuvent être définis à l'aide d'une liste de noms d'éléments et des caractères spéciaux pour indiquer où les raccourcis et les coches doivent être placés. C'est le type le plus commun de menu. LiveCode établira automatiquement les menus définis comme tels comme des menus utilisant l'apparence du système natif sur chaque plate-forme. Lorsque vous choisissez une option dans un menu de contenu, LiveCode enverra un message **menuPick** avec le nom de l'élément choisi.

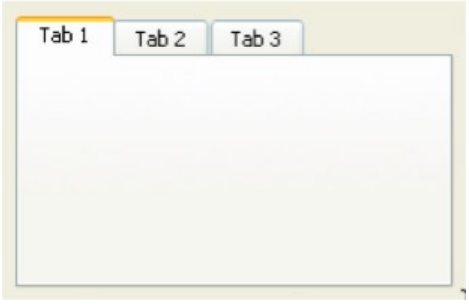
Alternativement, les menus peuvent être construits à partir d'un menu pile, vous donnant un contrôle complet sur son contenu et permettant l'affichage de tout type de fonctionnalité d'objet.
Lorsque vous choisissez un élément d'un menu pile, l'objet individuel dans le menu recevra un message **mouseUp**. Notez que les menus ne peuvent pas être affichés dans la barre de menu principale sur les systèmes Mac OS.

Pour plus de détails pour travailler avec les menus en général et les scripter, voir la section Utilisation des menus dans le chapitre Programmation d'une interface utilisateur.

Le menu en cascade est un type spécial de contrôle qui est utilisé seulement lors de la construction d'un menu du panneau de pile. Les éléments des sous-menus peuvent être créés dans les menus de contenu basés sur une liste sans utiliser cet objet.



Menu Controls



Tab Menu on Windows

Le panneau à onglets est un type de menu dans LiveCode. Vous pouvez spécifier une liste des onglets à afficher et recevoir un message **menuPick** quand l'utilisateur clique un onglet de la même manière que pour les autres menus.

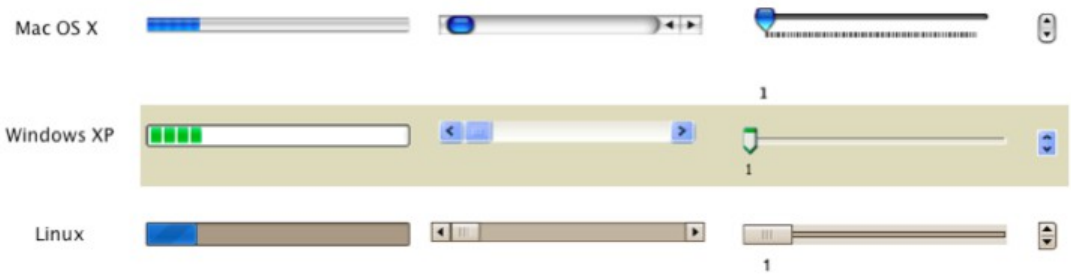
Il existe deux techniques communes pour la mise en œuvre d'une interface à onglets : regrouper les objets pour chaque onglet ensemble et afficher ou masquer le groupe approprié lorsque vous changez onglet, ou placez l'objet onglet, dans un groupe qui est ensuite placé en arrière-plan de plusieurs cartes.

4.2.29 Autres Contrôles

Les barres de défilement peuvent être utilisées comme une barre de progression pour afficher une valeur, un curseur pour permettre à l'utilisateur de choisir une valeur, ou pour faire défiler les objets.

Notez que vous n'avez pas besoin d'utiliser un objet de défilement avec des champs ou des groupes puisque ceux-ci peuvent afficher une barre de défilement incorporée.

Curseurs et ascenseurs peuvent être affichés à la fois horizontalement et verticalement - pour afficher verticalement, redimensionner afin que la hauteur soit supérieure à la largeur.

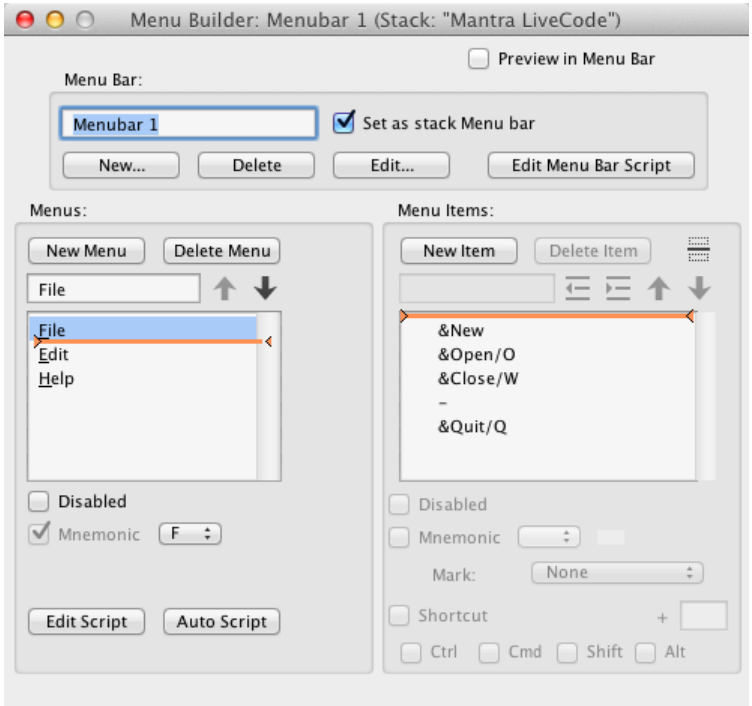


Scrollbars

4.3 Utilisation du Générateur de Menu (Menu Builder)

Le générateur de menu vous permet de créer et modifier une barre de menu standard qui fonctionne correctement indépendamment de la plate-forme pour laquelle vous avez l'intention de déployer votre application pour Windows et Unix, les menus créés avec le Générateur de menu apparaissent dans le haut de la fenêtre. Sur Mac OS, ils seront affichés dans la barre du menu principal. Il est également possible de générer une barre de menu par script. Pour plus de détails, voir la section sur la programmation d'une interface utilisateur.

Choisir **Tools > Menu Builder** pour ouvrir le générateur de menu.



Constructeur de Menus (Menu Builder)

Preview Cette option vous permet de prévisualiser votre barre de menu dans la barre du menu principal. Elle s'applique uniquement sur les systèmes Mac OS où l'option **Set as Menu Bar** sur Mac OS a été activée.

Important : Pour ramener la barre de menus développement de LiveCode lorsque vous travaillez avec une pile qui a cette option activée, cliquez sur une fenêtre de l'EDI LiveCode comme par exemple la barre d'outils.

Menu Bar Cette zone indique les principaux paramètres de votre barre de menu. Utilisez le bouton **New** pour créer une nouvelle barre de menu dans le haut de la pile courante. Entrez le nom de votre barre de menu dans la zone de texte. **Delete** permet de supprimer définitivement votre barre de menu. Utilisez le bouton **Edit** pour charger une barre de menu existante.

Menu edit area Sélectionner un menu pour travailler à partir de la liste déroulante. Au minimum, votre menu devrait avoir un Fichier, Edition et le menu Aide.

Ces menus sont créés pour vous automatiquement lorsque vous créez une nouvelle barre de menu.

Pour créer un nouveau menu, déplacez la barre de séparation orange sur la position du menu que vous souhaitez créer et appuyez sur **New Menu**.

Désactiver le menu actuellement sélectionné en validant **Disabled**.

Choisissez le raccourci clavier (la partie du nom qui est souligné), en utilisant le menu contextuel de **Mnemonic** (Windows, Linux et Unix uniquement).

Pour déplacer un menu dans la liste, sélectionnez-le et appuyez sur les flèches (à droite de la zone du nom) vers le haut ou vers le bas

Menu content area Sélectionnez un élément du menu pour travailler à partir de la liste déroulante.

Pour créer un nouvel élément de menu, déplacez la barre de séparation orange sur la position du menu que vous souhaitez créer et appuyez sur ***New Item***.

Choisissez le raccourci clavier (la partie du nom qui est souligné, à utiliser lorsque le menu est ouvert), en utilisant le menu contextuel de **Mnemonic** (Windows, Linux et Unix uniquement).

Pour déplacer un élément de menu vers le haut ou vers le bas de la liste, sélectionnez-le puis cliquez sur la flèche haut ou bas (à droite de la zone de nom).

Pour déplacer des éléments dans un sous-menu, cliquez sur la flèche droite ou cliquez sur la flèche gauche pour déplacer un élément de menu en arrière dans la barre du menu principal.

Pour insérer un diviseur, positionner la barre de séparation orange, où vous souhaitez le diviseur, cliquez sur le bouton bleu de division (en haut à droite).

Pour changer l'élément du menu en apparence diamant ou bien en case à cocher, choisissez l'option appropriée dans le menu contextuel **Mark**.

Pour créer un raccourci des touches de commande pour l'élément du menu, cliquez sur la case à cocher de raccourci et entrez la lettre que vous souhaitez utiliser pour le raccourci.

Pour comprendre les symboles qui sont créés à côté des éléments de menu, voir la section sur les barres de menus dans le chapitre sur la programmation d'une interface utilisateur.

Scripting Edit script ouvre l'éditeur de code pour le menu actuellement sélectionné.

Auto Script met un script vierge dans le bouton actuellement sélectionné avec un espace pour insérer les actions pour chacun des éléments de menu dans cet article.

Nous recommandons de cliquer sur **Auto Script** avant d'appuyer sur **Edit Script** lorsque vous créez un menu.

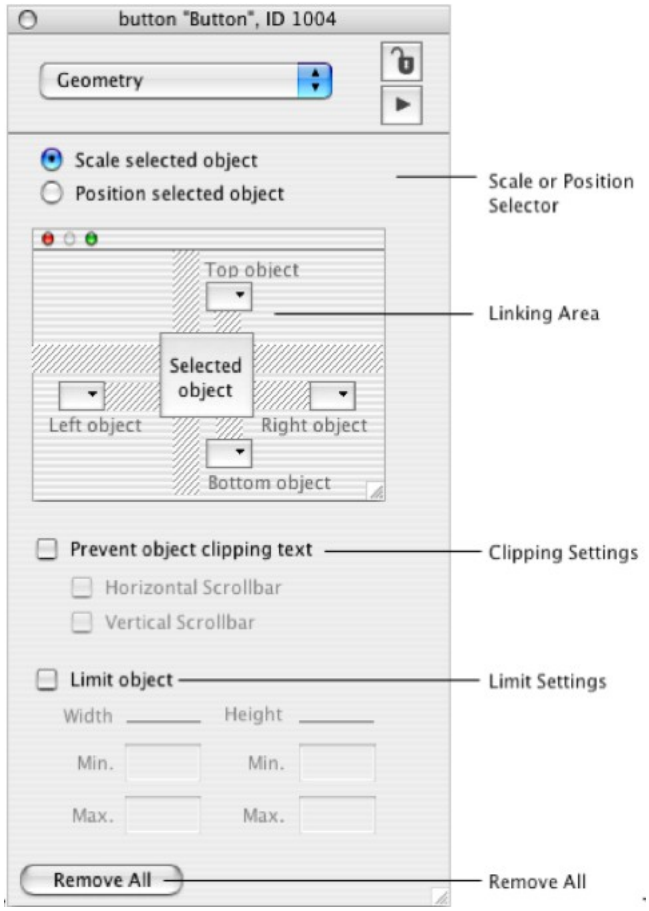
4.4 Utiliser le gestionnaire de géométrie (Geometry Manager)

Utilisez le gestionnaire de géométrie pour définir comment les objets doivent être mis à l'échelle et positionnés lorsque l'utilisateur redimensionne la fenêtre.

The Geometry Manager

Scale or Position Selector

Choisissez si vous souhaitez que le contrôle soit mis à l'échelle ou positionné lorsque la pile est redimensionnée. **Scale** modifie l'échelle du contrôle puisque la pile est redimensionnée. **Position** permet déplacer le contrôle, sans modifier ses dimensions. A noter qu'il est possible de mettre à l'échelle un objet dans le plan horizontal et qu'il soit sur **Position** dans le sens vertical. Sélectionnez **Scale** puis définissez les options pour un axe dans la zone de liaison. Ensuite, sélectionnez **Position** et définissez les options pour l'autre axe. Si vous définissez des options à la fois pour l'échelle et la position concernant les deux axes, les options d'échelle seront ignorés.



Linking area

Utilisez la zone de liaison pour spécifier la relation entre le contrôle et la fenêtre, ou d'autres contrôles lorsque la pile est redimensionnée. En mode de mise à l'échelle, vous pouvez lier chaque bord de l'objet à la fenêtre ou un autre objet. En mode de positionnement, vous pouvez lier les axes X et Y d'un objet. Cliquez sur barres grises pour créer un lien. Un simple clic entraîne un lien absolu, un second clic va créer un lien relatif. Un lien absolu permet de maintenir l'objet distant du nombre de pixels courant auquel il est lié. Par exemple, si vous liez le bord droit au bord de la fenêtre et que ce bord est actuellement à 10 pixels du bord de la fenêtre, chaque fois que la pile est

redimensionnée, le bord restera exactement éloigné de 10 pixels .
Toutefois, si vous utilisez un lien relatif la distance sera calculée comme un pourcentage de la largeur totale de la carte.
L'objet maintiendra le même pourcentage de distance par rapport au bord de la carte, le nombre exact de pixels variant alors.

Lors de la liaison à un autre contrôle, assurez-vous de créer un lien vers un contrôle qui est déplacé par rapport à la fenêtre, ou par un script (par exemple dans une commande de **resizeStack**).

Lorsque vous utilisez le gestionnaire de géométrie avec une commande de **resizeStack** existante assurez-vous de faire passer le message de **resizeStack**, sinon le gestionnaire de géométrie ne sera pas en mesure de prendre effet. Pour forcer la géométrie à se mettre à jour manuellement, utilisez **revUpdateGeometry**.

Astuce : Vous pouvez utiliser le gestionnaire de géométrie pour mettre à l'échelle des objets avec une barre de division. Créer et scripter la barre à déplacer, ensuite relier les bords des contrôles à elle, puis appeler **revUpdateGeometry** à chaque déplacement pour avoir une mise à échelle des objets automatique.

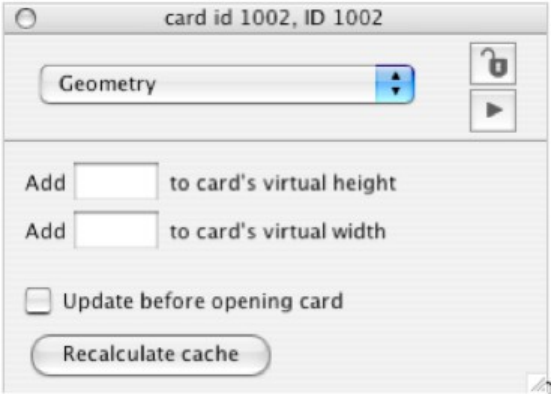
Clipping settings Activez ***Prevent Object Clipping Text*** pour empêcher que le contrôle devienne trop petit pour afficher son libellé quand la fenêtre est redimensionnée. Si le contrôle est un champ, vous pouvez également activer l'option pour afficher les barres de défilement si le texte dans le champ n'y contient pas.

Limit settings Permet de définir les minimum et maximum possibles pour les largeurs et hauteurs de l'objet.

Remove All Supprime tous les paramètres de géométrie de la commande. Utilisez cette option si les paramètres que vous avez appliqués ne donnent pas l'effet désiré et que vous voulez recommencer à zéro.

Les options **Geometry Card Settings** peuvent être consultées depuis l'inspecteur des propriétés des cartes. Utilisez ces options pour déterminer la façon dont la géométrie est appliquée sur les contrôles au sein de la carte actuelle.

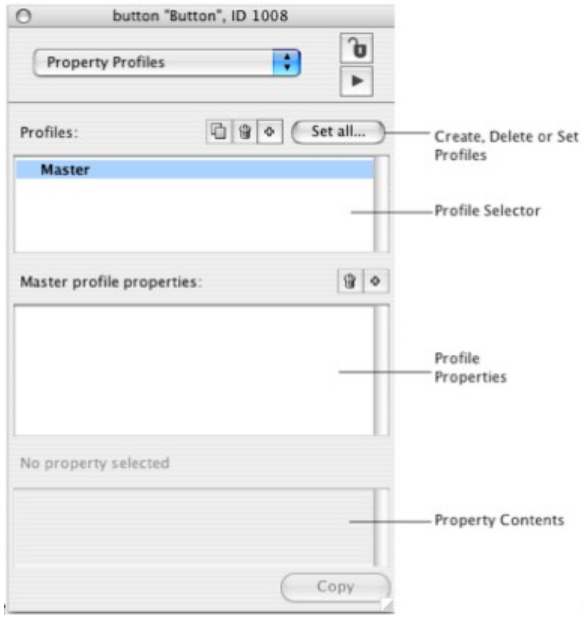
Réglages de la Géométrie de Carte



Add to cards virtual width or height Utilisez cette option pour mettre en œuvre une disposition qui permet à une partie des commandes optionnelles d'être déployées. Le gestionnaire de géométrie ignore la hauteur ou la largeur supplémentaire en pixels spécifiée dans cette zone, redimensionnant des objets comme si cette zone de la carte n'avait pas été élargie. Normalement, ces valeurs sont définies par le script lorsque la fenêtre est redimensionnée pour déployer des contrôles supplémentaires. Pour définir ces propriétés par le script, définissez les propriétés de la carte **cREVGeneral ["virtualWidth"]** ou **cREVGeneral ["virtualHeight"]**.

Update before opening card Provoque des redimensionnements des objets lors de la navigation sur la carte si la fenêtre a été redimensionnée depuis une autre carte. Cette option n'est pas nécessaire si les contrôles sont contenus dans un arrière-plan qui a déjà été redimensionné correctement aux dimensions actuelles de la fenêtre.

4.5 Utilisation des profils de propriété



Utilisez les profils de propriété pour stocker différents ensembles de propriétés en un seul objet. Les profils de la propriété peuvent être utilisés pour fournir des versions localisées de l'application, ou différents thèmes ou habillages.

Profils de propriété

Create, delete or set Profiles Les icônes de gauche à droite vous permettent de dupliquer, supprimer ou de créer un nouveau profil pour l'objet sélectionné. Le bouton **Set all** vous permet de définir tous les objets sur la carte ou la pile actuelle selon le profil actuellement sélectionné dans la zone de sélection d'un profil.

Lorsque vous créez un nouveau profil, assurez-vous de choisir un nom qui soit valide en tant que variable et ne soit pas un mot réservé de LiveCode. Utilisez des noms cohérents pour vous permettre de créer un thème ou la traduction et de définir tous les objets de votre carte ou pile du même profil.
Lorsque vous créez un nouveau profil, LiveCode commute automatiquement l'objet pour utiliser ce profil. Il y a deux façons d'inclure les nouveaux paramètres de propriété dans un profil : à l'aide de l'inspecteur des propriétés pour spécifier les propriétés que vous souhaitez inclure, et en modifiant les propriétés directement alors que le profil est actif.
Si vous apportez des modifications aux propriétés de l'objet, l'éditeur de profil permettra de suivre les changements et de sauver ceux-ci dans le profil actuel. Toutes les propriétés qui n'ont pas été définies pour le profil actuel seront héritées du profil principal (**Master**). Le système de profils prend en charge toutes les propriétés communes aux objets, y compris du texte stylé et des informations de géométrie. Toutefois, il ne stocke pas les propriétés qui se dupliquent l'une l'autre (par exemple, seule la valeur **rect** sera stockée, pas les propriétés d'emplacement à gauche, à droite ou autre, de l'objet). Les scripts et les propriétés personnalisées ne ont également pas stockées par l'éditeur de profil. Vous pouvez cependant écrire des scripts permettant d'abord de vérifier quel profil est utilisé par l'objet avant de lancer une action, en vérifiant la propriété de **revProfile** de l'objet.
Vous pouvez définir des profils par script en définissant la propriété **revProfile**. Pour définir la carte entière, la pile ou un fichier pile, utilisez les commandes respectivement **revSetcardProfile**, **revSetStackProfile** ou **revSetStackFileProfile**. Pour activer le stockage de nouvelles propriétés dans le profil actuel et donc modifier les profils plus rapidement basculer le **gRevProfileReadOnly** sur global.

Profile selector Sélectionnez un profil pour modifier toutes les propriétés de l'objet aux valeurs contenues dans ce profil. Cliquez sur le profil sélectionné pour mettre à jour la liste des propriétés stockées pour.

Profiles properties Affiche une liste de toutes les propriétés qui ont été modifiées dans le profil actuellement sélectionné, et ont donc une valeur unique sur ce profil. Sélectionnez une propriété pour voir son contenu. Appuyez sur l'icône plus pour ajouter manuellement une nouvelle propriété au profil actuel. Supprimer la propriété du profil actuel en utilisant l'icône de suppression.

Property contents Afficher et modifier le contenu de la propriété sélectionnée associée avec le profil sélectionné.

La boîte de dialogue Ajouter une propriété répertorie toutes les propriétés applicables, mais le volet ***Property Profiles*** élimine automatiquement les propriétés redondantes et synonymes de propriété.

Par exemple, si vous ajoutez les propriétés **backgroundColor** et **foregroundColor**, le volet ***Property Profiles*** de propriétés affiche la propriété couleurs à la place la prochaine fois que vous ouvrez l'inspecteur

des propriétés. C'est parce que la propriété couleurs contient tous les paramètres des huit couleurs d'un objet, il n'est donc pas nécessaire de stocker les propriétés des couleurs individuelles une fois qu'elles auront été réglées. Pour copier facilement une valeur de propriété d'un autre profil, cliquez sur le bouton **Copy** dans la section du bas et choisissez le profil que vous souhaitez copier.

Pour plus de détails sur l'utilisation des profils de propriété, voir la section sur les profils de propriétés dans le chapitre Programmation d'une interface utilisateur.

4.6 Dix Conseils pour une bonne conception de l'interface utilisateur

Si vous créez un utilitaire simple pour vous-même, une poignée d'autres personnes, ou comme un projet de recherche, la conception de l'interface est moins importante. Toutefois, si vous créez un logiciel pour un large groupe d'utilisateurs finaux, vous devriez prendre le temps de concevoir soigneusement l'interface utilisateur. Les logiciels informatiques sont devenus plus matures au cours des deux dernières décennies, les attentes des utilisateurs de ce que leur expérience du logiciel devrait être ont augmenté. De plus en plus, les utilisateurs s'attendent, à des interfaces claires, concises, qui soient visuellement attrayantes. Bien faire les choses est pour partie un art et pour partie une science. Il est au-delà de la portée de ce manuel de donner des instructions détaillées sur ce sujet. Il y a beaucoup de bons livres et les ressources consacrés à ce domaine en constante évolution. Mais nous avons pensé qu'il serait utile de vous donner nos 10 meilleurs conseils pour une bonne conception de l'interface utilisateur.

Moins cest plus Ne pas utiliser trois boutons où un seul est nécessaire. Moins a de choix à faire un utilisateur, plus votre logiciel sera aisé à apprendre.

Un Design pour communiquer Un bon design appuiera tout ce que vous essayez de communiquer.

Obtenez les valeurs par défaut utiles Fournir des préférences est idéal pour les utilisateurs de chevronés. N'oubliez pas que la majorité de vos utilisateurs n'ira jamais ajuster les paramètres par défaut. Donc concentrez-vous sur la définition des comportements par défaut utiles avant de commencer à ajouter des préférences.

Mise en page Un environnement propre, cohérent permet de transmettre un sentiment de professionnalisme et rend votre logiciel utilisable. Si vous avez utilisé un bouton qui se trouve à 20 pixels de large dans une partie de votre interface alors utilisez la même taille de bouton ailleurs. Alignez vos objets avec soin.

Résolution d'Ecran Réfléchissez la résolution d'écran nécessaire et, si oui ou non, vous voulez que votre interface soit redimensionnable. Cette décision aura un grand impact sur votre conception.

Considérez le déroulement du programme Si votre programme effectue une tâche complexe avec de nombreuses options, envisagez de créer une interface de style assistant qui guidera l'utilisateur étape par étape dans sa tâche. De cette façon, il ne sera jamais confronté à un écran recouvert de dizaines d'options sans aucune idée de ce qu'il faut faire ensuite.

Tester sur des utilisateurs réels Sélectionnez un groupe de personnes et regardez-les utiliser le programme. Ne pas interférer ou les aider, simplement les laisser faire et prendre des notes. Si vous n'avez pas de budget pour les laboratoires de tests utilisateurs coûteux, ce processus peut être aussi simple que de réunir un groupe d'étudiants et de leur offrir une pizza gratuite. Vous obtiendrez une tonne de commentaires utiles qui va vraiment aider à rendre votre programme facile d'utilisation.

Ne pas utiliser des roues carrées Les différents types de widgets et de types de commandes disponibles ont acquis pour les utilisateurs un sens au cours des années d'utilisation. Ne pas utiliser quelque chose qui a une fonction déjà connue pour effectuer un autre type de tâche. Si vous avez besoin d'un widget qui fait quelque chose de nouveau, construisez quelque chose de nouveau.

Habillage Si vous produisez un habillage personnalisé pour votre application, une bonne règle de base est soit de le rendre entièrement personnalisé ou bien utiliser des widgets OS standard. Un bouton natif OS peut paraître très à sa place au le milieu d'un habillage (**skin**) personnalisé conçu avec soin.

Directives HCI pour les trois principales plates-formes Chacune des plates-formes, que LiveCode prend en charge, a son propre ensemble de lignes directrices pour son l'interface utilisateur. Nous vous recommandons de prendre le temps de vous familiariser avec.

- OS X Principes d'Interface Humaine
- iOS Design Applications
- Windows Principes d'Expérience Utilisateur
- GNOME Principes d'Interfaces Humaines

Chapitre 5 Coder avec LiveCode

Ecrire du code c'est savoir comment donner des fonctionnalités à une application. L'écriture d'un bon code signifie que votre application va faire ce que vous voulez qu'elle fasse. Heureusement, LiveCode qui est construit dans un langage de haut niveau, rend cette tâche facile. La syntaxe type anglais est facile à lire et à écrire. Ce chapitre vous guide à travers l'écriture en LiveCode.

5.1 La Structure d'un Script

5.1.1 Qu'est-ce qu'un Script

Chaque objet dans en LiveCode peut contenir un script qui lui dit quoi faire. Vous pouvez éditer le script d'un objet en utilisant l'éditeur de code. Un script est organisé en un ensemble de gestionnaires de messages individuels, dont chacun peut réagir à un événement différent (voir **Messages**) ou contenir un morceau de fonctionnalité spécifique. Les scripts peuvent envoyer des messages les uns aux autres. Dans une application bien organisée un script passera régulièrement des données entre les différents types de gestionnaires de messages qui ont été regroupés et organisés pour offrir des fonctionnalités, avec un minimum de double emploi. Un script peut contenir des sections de code commentées - remarques qui sont destinés aux humains et ne sont pas exécutées. Techniquement, un script est simplement une autre propriété de l'objet, ainsi un script peut en piloter un autre - dans certaines limites.

5.1.2 Les Types de Gestionnaires

Un gestionnaire est une section complète du code. Chaque gestionnaire peut être exécuté indépendamment. Il existe quatre types de gestionnaire : les commandes (gestionnaires de messages), les fonctions, les gestionnaires **GetProp** et les gestionnaires **setProp**.

5.1.3 Gestionnaires de Message

Chaque gestionnaire de message commence par la structure de contrôle suivie du nom du message qui répond à ce gestionnaire. Le gestionnaire se termine par la structure de commande de fin, et le nom du message.

Les gestionnaires de messages ressemblent à ceci :

```
on mouseUp
  beep
end mouseUp
```

Un gestionnaire de messages est exécuté lorsque l'objet dont le script contient le gestionnaire reçoit le message. Cet exemple gestionnaire répond au message **mouseUp** (voir **Messages**).

5.1.4 Gestionnaires de Fonction

Chaque gestionnaire de fonction commence avec la structure de commande de fonction suivie du nom de la fonction que calcule ce gestionnaire. Le gestionnaire se termine par la structure de commande de fin, et le nom de la fonction. Les gestionnaires de fonction sont généralement appelés par un autre gestionnaire et renvoient une valeur en utilisant la structure de contrôle de retour. Les gestionnaires de la fonction ressemblent à ceci :

```
function currentDay
  return item 1 of the long date
end currentDay
```

Un gestionnaire de fonction est exécuté quand un gestionnaire dans le même script (ou un dans un objet plus en dessous dans la hiérarchie de message) appelle la fonction. Cet exemple retourne le nom du jour d'aujourd'hui.

5.1.5 Gestionnaire GetProp

Chaque gestionnaire **getProp** commence par la structure de commande **getProp** suivie du nom de la propriété personnalisée auquel ce gestionnaire correspond. Le gestionnaire se termine par la structure de commande de fin, et le nom de la propriété. Les gestionnaires **getProp** ressemblent à ceci :

```
getProp myCustomProperty
  return the scroll of me + 20
end myCustomProperty
```

Un gestionnaire **getProp** est exécuté chaque fois que la valeur de la propriété personnalisée correspondante est demandée par une déclaration de LiveCode. Vous pouvez écrire un gestionnaire **getProp** pour une propriété personnalisée de l'objet ou d'un autre objet plus bas dans la hiérarchie du message. Pour plus d'informations, consultez la section Propriétés personnalisées.

5.1.6 Gestionnaire SetProp

Chaque gestionnaire **setProp** commence par la structure de commande **setProp** suivie du nom de la propriété personnalisée auquel ce gestionnaire correspond. Le gestionnaire se termine par la structure de commande de fin, et le nom de la propriété. Un gestionnaire **setProp** ressemble à ceci :

```
setProp myCustomProperty newSetting
  set the hilite of me to true
  pass myCustomProperty
end myCustomProperty
```

Un gestionnaire **setProp** est exécuté chaque fois que la valeur de la propriété personnalisée correspondante est modifiée par la commande **set**. Vous pouvez écrire un gestionnaire **setProp** pour une propriété personnalisée de l'objet ou d'un autre objet plus bas dans la hiérarchie d'objets. Pour plus d'informations, consultez la section Propriétés personnalisées.

5.1.7 Commentaires

Les commentaires sont des remarques qui sont destinées aux humains et ne sont pas exécutables. Pour certaines recommandations sur le type de commentaires à inclure, et quand, voir la section sur les bons

conseils de conception. Les commentaires peuvent être placés dans un gestionnaire, ou en dehors de tout gestionnaire.
Toute ligne (ou une partie d'une ligne) qui commence par deux tirets (--) ou un signe dièse (#) est un commentaire. Placer ces caractères au début d'une ligne se dit commenter (**commenting out**) une ligne.

```
on mouseUp -- ceci est un commentaire
  beep
  -- et ceci aussi...
end mouseUp
```

Vous pouvez supprimer temporairement une déclaration, ou même un gestionnaire entier, en le commentant. Pour commenter plusieurs lignes à la fois, sélectionnez-les et choisissez **Script > Comment**.
Puisque les commentaires ne sont pas exécutés, vous pouvez les placer n'importe où dans un script - à l'intérieur d'un gestionnaire ou à l'extérieur tous les gestionnaires.
Les commentaires qui commencent par - ou # s'étendent seulement jusqu'à la fin de la ligne.
Pour créer un commentaire sur plusieurs lignes, ou créer un bloc de commentaire, les entourer avec **/ *** et *** /** :

```
on openCard
  /* Ceci est en commentaire multi-ligne pouvant
contenir des informations détaillées sur ce gestionnaire,
telles que le nom de l'auteur, une description, ou la
date à laquelle le gestionnaire de la dernière modification. */
 show image "My Image"
  pass openCard /* On peut aussi l'utiliser sur une seule ligne */
end openCard
```

Les blocs de commentaires sont utiles lorsque vous souhaitez supprimer temporairement une section de code pendant le débogage d'un gestionnaire. Vous pouvez placer les caractères "/ *" au début de la section, et "* /" à la fin pour empêcher la section prévue de s'exécuter.

5.1.8 Compilation d'un script

Un script est compilé lorsque vous validez le script en cliquant sur le bouton **Apply** dans l'éditeur de code (ou, si modification d'un script à partir d'un autre script, en utilisant la commande **set**). Lors de la compilation, le script entier est analysé. Si une erreur de compilation se trouve lorsqu'un script est compilé, le script entier est indisponible pour l'exécution jusqu'à ce que l'erreur soit corrigée et que le script soit recompilé. Cela s'applique uniquement à des erreurs de compilation. Si une erreur d'exécution survient pendant l'exécution d'un gestionnaire, il n'affecte pas la possibilité d'utiliser d'autres gestionnaires dans le même script. Pour plus d'informations sur le traitement des erreurs, voir la section sur le débogage. Vous ne pouvez pas modifier un script pendant que son gestionnaire s'éxécute, parce que ce qui est exécuté est la version compilée, non le texte dans le script.

5.1.9 Résumé

Chaque objet a un script, qui peut être vide ou peut contenir un ou plusieurs gestionnaires de LiveCode. Vous modifiez un script en utilisant l'éditeur de code, ou en définissant la propriété de script de l'objet. Un script peut contenir quatre types de gestionnaires : les commandes, les gestionnaires de fonction, les gestionnaires **setProp** et gestionnaires **getProp**. Un commentaire est une partie du script qui n'est pas exécuté. Les commentaires commencent avec - ou #. Si un script contient une erreur de compilation, aucun de ses gestionnaires ne peut être utilisé jusqu'à ce que l'erreur soit corrigée.

5.2 Evénements

LiveCode est basé sur des événements. Chaque action qu'un script génère est déclenchée par un événement

qui est envoyé sous la forme d'un message.

5.2.1 Qu'est-ce qui provoque l'Envoi des Messages

Les messages sont envoyés par des événements. Ces événements incluent les actions des utilisateurs (tels que de frapper sur une touche ou cliquer sur un bouton de la souris) et les actions du programme (comme terminer un téléchargement de fichier ou quitter l'application).
LiveCode attend des événements et envoie un message à l'objet approprié lorsqu'un événement se produit. Ces messages sont dénommés messages intégrés, et incluent **mouseDown**, **mouseUp**, **keyDown**, **openCard**, et tous les autres messages décrits dans le dictionnaire de LiveCode. LiveCode envoie également un message chaque fois qu'un gestionnaire exécute une commande personnalisée (voir Envoi de messages). Toutefois, les commandes intégrées sont exécutées directement par le moteur et ne donnent pas lieu à l'envoi d'un message.
De même, LiveCode envoie un appel de fonction à chaque fois qu'un gestionnaire appelle une fonction personnalisée, un déclencheur de **setProp** chaque fois qu'un gestionnaire définit une propriété personnalisée, et un appel **getProp** chaque fois un gestionnaire obtient la valeur d'une propriété personnalisée.

5.2.2 Répondre aux événements

Pour répondre à un message, vous écrivez un gestionnaire de messages avec le même nom que le message. Par exemple, pour répondre à un message **keyDown** envoyé à un champ (qui est envoyé lorsque l'utilisateur appuie sur une touche alors que le point d'insertion est dans le champ), placez un gestionnaire **keyDown** dans le script du champ :

```
on keyDown theKey -- répond à un message keyDown
  if theKey is a number then beep
end keyDown
```

Note : Quand un outil autre que l'outil ***Browse*** est activé, l'environnement de développement bloque les messages intégrés, qui sont normalement envoyés en cliquant (tels que **mouseDown** et **mouseUp**). C'est ainsi que, par exemple, vous pouvez utiliser l'outil ***Pointeur*** pour sélectionner et déplacer un bouton sans déclencher son gestionnaire **mouseUp**.

5.3 Le Chemin de Message

Le chemin du message est l'ensemble des règles qui déterminent quels objets, dans quel ordre, ont la possibilité de répondre à un message. Le chemin de message est basé sur la hiérarchie d'objets.

5.3.1 Hiérarchie des Objets

Chaque objet LiveCode fait partie d'un autre objet, d'un type différent. Par exemple, chaque carte fait partie d'une pile, chaque commande groupée fait partie d'un groupe, et ainsi de suite. Cette hiérarchie d'objets définit la propriété et la relation d'héritage entre objets.

Les propriétés de police, de couleur et de modèle, sont héritées du propriétaire de l'objet si elles ne sont pas définies à leur tour. Cela signifie que si vous définissez le **textFont** d'une pile, tous les objets de cette pile qui n'ont pas leur propre jeu de propriétés **textFont** vont utiliser cette police de texte.

5.3.2 Le Chemin de Message

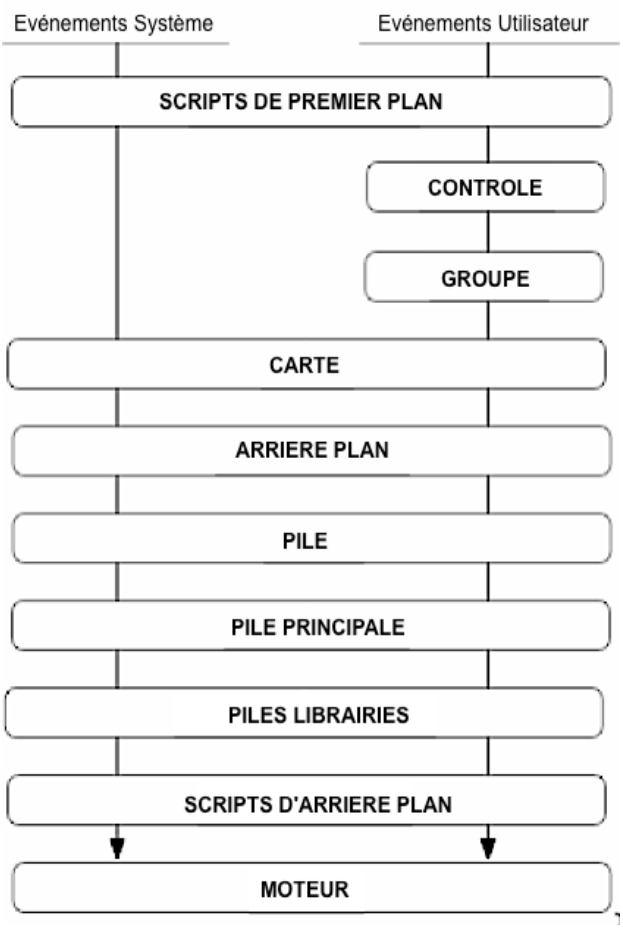
Quand un message est envoyé à un objet, il est souvent géré directement par un gestionnaire de messages dans cet objet même. Toutefois, si aucun gestionnaire n'est présent, le message continuera le long du chemin jusqu'à ce qu'il trouve un gestionnaire de messages pouvant y répondre. Cela permet de regrouper ensemble des fonctionnalités similaires à différents niveaux au sein de votre application. Ce comportement s'applique à la fois aux messages d'événement envoyés à la suite d'une action de l'utilisateur et aux messages

personnalisés envoyés par script. Il est donc possible d'écrire des bibliothèques de fonctions communes.

La hiérarchie de l'objet est étroitement lié au chemin dans lequel se déplace un message. Dans la plupart des cas, lorsqu'un objet passe un message, le message va au propriétaire de l'objet dans la hiérarchie d'objets.

Le chemin du message est détaillé dans la figure ci-contre.

Le chemin du message



Par exemple, supposons que l'utilisateur clique sur un bouton (contrôle) dans une pile principale, forçant LiveCode à envoyer un message **mouseUp** au bouton. Si le script du bouton ne contient pas de gestionnaire pour le message **mouseUp**, le message est transmis à la carte, tant que le bouton est activé.

Si le script de la carte contient un gestionnaire **mouseUp**, le gestionnaire est exécuté. Mais si la carte ne gère pas le message **mouseUp**, il est transmis à la pile de la carte.

Si le script de la pile contient un gestionnaire **mouseUp**, le gestionnaire est exécuté. Mais si la pile ne gère pas le message **mouseUp**, elle est transmise au moteur.

Le moteur est le bout du chemin du message, et si un message lui parvient, le moteur prend toute action par défaut (par exemple, l'insertion d'un caractère dans un champ ou souligner un bouton), puis jette le message.

Si un message correspondant à une commande personnalisée ou un appel de fonction personnalisée atteint la fin du chemin de message sans trouver aucun gestionnaire, au lieu d'être jeté, il provoque une erreur d'exécution.

Note: Pour être considéré comme un arrière-plan (selon le schéma du chemin de message ci-dessus), un groupe doit avoir sa propriété **backgroundBehavior** réglé sur vrai.

5.3.3 La Cible du Message

L'objet à qui un message a été envoyé à l'origine, est appelé la cible du message. Vous pouvez obtenir la cible depuis n'importe quel gestionnaire dans le chemin du message en utilisant la fonction cible. Par exemple, si vous cliquez sur un bouton (provoquant la diffusion d'un message **mouseUp**), et le script du bouton contient un gestionnaire **mouseUp**, alors la fonction de cible retourne le nom du bouton. Toutefois, si le bouton ne gère pas le message **mouseUp**, ce dernier est passé à la carte, et si la carte dispose d'un gestionnaire **mouseUp**, il est exécuté en réponse au message. Dans ce cas, le script de la carte est en cours d'exécution, mais l'objectif n'est pas la carte - c'est la touche qui a été initialement cliquée, parce que LiveCode envoyé le message **mouseUp** au bouton.

Astuce : Pour obtenir le nom de l'objet dont le script est en cours d'exécution, utilisez le mot clé me.

5.3.4 Gestionnaires avec le même nom

Si deux objets différents dans le chemin du message ont chacun un gestionnaire avec le même nom, le message est traité par le premier objet qui le reçoit et dispose d'un gestionnaire pour lui. Par exemple, supposons que le script d'un bouton intègre un gestionnaire **mouseUp**, tout comme le script de la pile. Si vous cliquez sur le bouton, un message **mouseUp** est envoyé au bouton. Parce que le script du bouton a un gestionnaire **mouseUp**, que le bouton gère ce message, il n'ira pas plus loin. Le message n'est jamais envoyé au script de la pile, de sorte que pour cet événement de clic, le gestionnaire **mouseUp** du script de la pile ne sera pas exécuté.

Note : Si le script d'un objet a plus d'un gestionnaire avec le même nom, le premier est exécuté lorsque l'objet reçoit le message correspondant. Les autres gestionnaires dans le script du même objet avec le même nom ne sont jamais exécutés.

5.3.5 Piéger les messages

Lorsqu'un objet reçoit un message et qu'un gestionnaire pour ce message est trouvé, le gestionnaire est exécuté. Normalement, un message qui a été traité ne va pas plus loin le long du chemin du message. Son objectif ayant été atteint, il disparaît. Un tel message est dit avoir été piégé par le gestionnaire.

Si vous voulez éviter à un message d'être poussé plus loin sur le chemin du message, mais ne souhaitez pas faire quelque chose avec, un gestionnaire vide pour le message bloquera sa transmission. Cet exemple empêche que le message **mouseDown** soit sollicité par l'objet, le gestionnaire étant vide, ainsi que tous les objets situés plus bas dans la hiérarchie d'objets :

```
on mouseDown
  -- Gestionnaire vide
end mouseDown
```

Vous pouvez utiliser la même technique pour bloquer les appels de fonctions personnalisées, les déclencheurs **setProp**, et des appels **getProp**.

5.3.6 Bloquer les Messages Systèmes

Vous pouvez bloquer les messages du système - par exemple ceux envoyés lorsque vous naviguez vers une autre carte - d'être envoyés, alors qu'un gestionnaire est en cours d'exécution en définissant la propriété **lockMessages** sur vrai.

Par exemple, si le gestionnaire ouvre une autre pile, LiveCode envoie normalement les messages **openCard** et **openStack** à la pile. Si la pile contient des gestionnaires de ces messages risquant d'entrainer des comportements indésirables au cours de cette opération, vous pouvez utiliser la commande **lockMessages** avant d'aller à la pile afin de bloquer temporairement ces messages. Lorsque le gestionnaire a fini de s'exécuter, la propriété **lockMessages**, revient automatiquement à sa valeur par défaut, sur faux, et l'envoi des messages normaux reprend.

Astuce : Pour bloquer les messages de navigation tout en testant ou débogant une pile, appuyez sur **Suppress Message** dans la barre d'outils ou choisissez **Development > Suppress Messages.**

5.3.7 Passage d'un Message à l'Objet Suivant

Pour laisser un message aller plus loin sur le chemin du message, utilisez la structure de contrôle **Pass**. La structure de contrôle **Pass** arrête le gestionnaire actuel et envoie le message à l'objet suivant dans le chemin du message, comme si l'objet ne contenait pas de gestionnaire pour le message :

```
on openCard
  executerLeProgrammePrevuPourCetteCarte
  pass openCard – laisser la pile recevoir aussi le message
end openCard
```

5.3.8 Piégeage sélectif ou Transmettre des Messages

Des messages intégrés, tels que **keyDown**, déclenchent une action, de sorte que le piégeage du message empêche l'action d'être exécutée . L'exemple suivant passe le message **keyDown** que si le caractère tapé est un nombre :

```
on keyDown theKey
  if theKey is a number then pass keyDown
end keyDown
```

Si vous placez ce gestionnaire dans le script d'un champ, il permet à l'utilisateur de saisir des chiffres, mais d'autres combinaisons de touches sont piégées par le gestionnaire **keyDown** et non autorisées à passer.

Un principe similaire s'applique aux déclencheurs de **setProp**. Si un gestionnaire de **setProp** ne passe pas le déclencheur du **setProp**, la propriété personnalisée n'est pas définie.

5.3.9 Groupes, Arrière-plan et Chemin de Message

Comme vous pouvez le voir sur le schéma ci-dessus la position d'un groupe dans le chemin d'un message dépend de savoir si la case à cocher Comportez-vous comme arrière-plan (***Behave as Background***) a été activée (par script ou en utilisant la propriété **backgroundBehavior**).

Si le comportement de **Background** est sur faux, le groupe est dans le chemin de message pour tous les contrôles qu'il détient, mais n'est pas dans le chemin du message d'un autre objet. Si le comportement de **Background** est sur vrai, le groupe est également dans le trajet des messages pour toutes les cartes dans lesquelles il est placé. Si vous envoyez un message à un contrôle de la carte, le message passe par le contrôle, la carte, puis des groupes d' arrière-plan sur la carte dans l'ordre des numéros, puis la pile.

Puisque un groupe possède tous les contrôles qui font partie du groupe, si vous envoyez un message à un contrôle au sein d'un groupe, le groupe est dans le chemin de message pour ses propres contrôles, indépendamment du fait que le comportement d'arrière-plan est sur vrai ou faux. Si un groupe a déjà reçu un message, car il a été envoyé à l'une des commandes du groupe, le message n'est pas envoyé par le groupe à nouveau après que la carte l'ait traité.

5.4 Commandes et Fonctions

Vous utilisez des commandes et des fonctions pour exécuter les actions de votre application. Les commandes instruisent la demande de faire quelque chose comme lire un film, afficher une fenêtre, ou modifier une propriété. Les fonctions calculent une valeur. Différentes fonctions peuvent ajouter une colonne de chiffres, ou obtenir la quinzième ligne d'un certain fichier ou savoir si une touche est enfoncée.

5.4.1 Utiliser les Commandes et Fonctions Intégrées

LiveCode a plus de cent cinquante commandes intégrées, et plus de deux cents fonctions intégrées, qui sont documentés dans le dictionnaire de LiveCode.

Commande

Une commande est une instruction pour LiveCode de faire quelque chose. Une commande est placée au début d'une instruction (soit le début d'une ligne ou après une structure de contrôle tels que **then**). La

commande est suivie par tous les paramètres qui spécifient les détails de ce que la commande à faire.

Voici quelques exemples de la façon dont des commandes intégrées sont utilisées en l'état :

go **next** card – **commande go**
beep **-- *commande* beep**
set the hilite of me to true **-- *commande* set**

Fonctions

Un appel de fonction est une demande de LiveCode pour information. Une fonction est déclarée en utilisant le nom de la fonction, puis en ouvrant et fermant parenthèses qui peuvent contenir des paramètres qui spécifient les détails de ce que la fonction doit calculer.

Lorsque vous utilisez une fonction dans une déclaration, LiveCode appelle la fonction pour calculer les informations spécifiées, puis substitue cette information dans le script comme si le scénario avait initialement écrit cette information à la place de l'appel de fonction. Les informations renvoyées peuvent donc être mises dans une variable ou autre conteneur en utilisant la commande **put** devant une fonction.

Voici un exemple de la façon dont une fonction est utilisée :

put round(22.3) into field "Number"

Lorsque cette instruction est exécutée, LiveCode appelle la fonction round. Quand on arrondit 22,3, le nombre résultant est de 22, alors la déclaration met le numéro 22 dans le champ.

Le nombre que vous êtes en train d'arrondir est placé entre parenthèses après le nom de la fonction round. Ce nombre est appelé le paramètre de la fonction. Une fonction peut avoir un paramètre, ou aucun, ou plusieurs. Les paramètres sont entre parenthèses et, s'il y a plus d'un, ils sont séparés par des virgules.

Voici quelques exemples :

put date() into myVariable **-- fonction de date, pas de paramètres passés**
length("hello") into me **-- fonction de longueur, 1 paramètre passé**
average(10,2,4) **-- Fonction moyenne, 3 paramètres passés**

Important : Un appel de fonction, en soi, n'est pas une déclaration complète. Il faut utiliser un appel de fonction en combinaison avec une structure de commande ou de contrôle. (Dans le premier exemple ci-dessus, la fonction d'arrondi est utilisée avec la commande **put**).

Ecrire des appel de fonction avec la forme *the*

Si une fonction intégrée n'a pas de paramètres ou un seul paramètre, il peut être écrit sous une forme différente, sans parenthèses :

put the date into myVariable **--fonction date**
put the length of "hello" into me **--fonction longueur**

Si la fonction n'a pas de paramètres, cette forme s' écrit comme **the functionName**. Si elle a un paramètre,

elle s'écrit comme **the functionName of** son paramètre.

La forme "**the**" fonctionne de la même manière que la forme «**( )**» ci-dessus, et vous pouvez utiliser les deux formes indifféremment pour des fonctions intégrées avec moins de deux paramètres. L'entrée du dictionnaire de LiveCode pour chaque fonction intégrée montre comment écrire les deux formes.

Important : Vous pouvez utiliser la forme **the** pour les fonctions intégrées, mais pas pour les fonctions personnalisées que vous écrivez. L'écriture des fonctions personnalisées est discutée plus loin.

5.4.2 Les commandes et les fonctions personnalisées

Vous utilisez des commandes personnalisées et des fonctions personnalisées de la même manière que toute autre commande ou une fonction.

Utilisation des commandes personnalisées

Vous pouvez exécuter une commande personnalisée en tapant simplement le nom de la commande que vous souhaitez envoyer.

```
MyCommand
```

Vous répondez au message d'une commande personnalisée de la même manière, en écrivant un gestionnaire de messages avec the nom de la commande :

```
on myCommand -- a custom command
  beep 3
  go next card
  add 1 to field "Card Number"
end myCommand
```

Si vous ne spécifiez pas un objet, le message est envoyé à l'objet dont le script en cours d'exécution, et passe ensuite dans la hiérarchie de message comme d'habitude.

Comme une commande intégrée, une commande personnalisée est une instruction pour LiveCode de faire quelque chose. Vous pouvez inclure des paramètres avec une commande personnalisée en les passant à la suite du nom :

```
checkForConnection "ftp://ftp.example.org"
makePrefsFile fileLoc,field "Preferences"
```

Les commandes intégrées peuvent avoir une syntaxe très souple :

```
go to card 3 of stack "Dialogs" as modal
group image "Help Icon" and field "Help Text"
hide button ID 22 with visual effect dissolve very fast
```

Toutefois, la syntaxe des commandes personnalisées est plus limitée.
Une commande personnalisée peut avoir plusieurs paramètres, et s'il en existe plus d'un, ils sont séparés par des virgules :

```
libURLDownloadToFile myFile,newPath,"downloadComplete"
revExecuteSQL myDatabaseID,field "Query","*b" & "myvar"
```

Mais les commandes personnalisées ne peuvent pas utiliser des mots comme **and** et **to** pour réunir les fragments de la commande, à la façon des commandes intégrées. Pour cette raison, les commandes personnalisées ne peuvent ne pas être aussi proche de l'anglais que des commandes intégrées peuvent l'être.

Utilisation des fonctions personnalisées

Lorsque vous utilisez la fonction personnalisée "**fileHeader**" dans une déclaration, le gestionnaire de la fonction est exécutée, et l'appel de fonction est remplacé par la valeur de l'instruction en retour. Cet exemple montre comment la fonction peut être utilisée :

```
put fileHeader(it) into myFileHeaderVar
```

Et la fonction personnalisée correspondante :

```
function fileHeader theFile
  put char 1 to 8 of URL ("file:" & theFile) into tempVar
  put binaryDecode("h*",tempVar) into tempVar
  return tempVar
end fileHeader
```

Comme une fonction intégrée ou une fonction bibliothèque, un appel de fonction personnalisée est une demande de renseignements. Voici quelques exemples montrant l'utilisation des fonctions personnalisées fabriquées :

```
get formattedPath("/Disk/Folder/File.txt")
put summaryText(field "Input") into field "Summary"
if handlerNames(myScript, "getProp") is empty then beep
```

Les fonctions personnalisées n'ont pas de forme **the**, et sont toujours écrites avec parenthèses après la nom de la fonction. Si la fonction a des paramètres, ils sont placés à l'intérieur des parenthèses, séparés par des virgules. Si la fonction n'a pas de paramètres, les parenthèses sont vides.

5.4.3 Passer des Paramètres

Une valeur que vous passez d'un gestionnaire à l'autre s'appelle un paramètre. Dans l'exemple ci-dessous, la déclaration suivante envoie le message **alertUser** avec un paramètre :

```
alertUser "Vous avez cliqué sur un bouton !"
```

Le paramètre "Vous avez cliqué sur un bouton !" est passé au gestionnaire **alertUser**, qui accepte le paramètre et le place dans une variable de paramètre appelé **theMessage**. Le gestionnaire **alertUser** peut ensuite utiliser le paramètre dans ses déclarations :

```
on alertUser theMessage
  beep
  answer theMessage – utilise le parmètre "theMessage"
end alertUser
```

5.4.4 Passer de Multiples Paramètres

Si une déclaration passe plus d'un paramètre, les paramètres sont séparés par des virgules. L'exemple suivant a deux paramètres, "**theMessage**" et "**numberOfBeeps**" :

```
on seriouslyBugUser theMessage,numberOfBeeps
  beep numberOfBeeps
  answer theMessage
end seriouslyBugUser
```

Pour utiliser cette commande personnalisée, vous l'appelez comme ceci :

```
seriouslyBugUser "Bonjour",5
```

Lorsque le gestionnaire "**seriouslyBugUser**" est exécuté avec les déclarations ci-dessus, le paramètre **theMessage** est "Bonjour" et le paramètre **numberOfBeeps** est 5.

5.4.5 Variables de paramètres

Dans l'exemple ci-dessus, "**theMessage**" et "**numberOfBeeps**" sont les variables de paramètres. Vous déclarez des variables de paramètres dans la première ligne d'un gestionnaire. Lorsque le gestionnaire commence à s'exécuter, les valeurs que vous avez passé sont placées dans les variables de paramètres.

Vous pouvez utiliser des variables de paramètres de la même façon que les variables ordinaires : vous pouvez y placer des données, les utiliser dans les expressions, et ainsi de suite.
Variables de paramètres sont des variables locales, de sorte qu'elles n'ont plus d'existence dès que le gestionnaire arrête son exécution.

Les noms des variables de paramètres
Vous pouvez donner un nom qui est un nom de variable légal. Les noms de variables doivent être constitués d'un seul mot et peuvent contenir n'importe quelle combinaison de lettres, chiffres et caractères de soulignement _ . Le premier caractère doit être une lettre ou un trait de soulignement. Ce n'est pas le nom, mais l'ordre des paramètres est important.

Paramètres Vides
Un gestionnaire peut avoir n'importe quel nombre de variables de paramètres indépendamment du nombre de paramètres qui lui sont passés. S'il y a plus de variables de paramètres qu'il y a de valeurs à mettre en eux, les variables de paramètres restantes demeurent vides. Considérons le gestionnaire suivant, qui a trois variables de paramètres :

```
on processOrder itemName,itemSize,numberOfItems
  put itemName into field "Name"
  put itemSize into field "Size"
  if numberOfItems is empty then put 1 into field "Number"
  else put numberOfItems into field "Number"
end processOrder
```

La déclaration suivante passe un ordre pour un pull :

```
processOrder "sweater","large"
```

La déclaration ne transmet que deux paramètres, tandis que le gestionnaire "**processOrder**" a trois variables de paramètres, de sorte que la troisième variable paramètre, "**numberOfItems**", est vide. Parce que le

gestionnaire prévoit la possibilité que «**numberOfItems**" puisse être vide, vous pouvez passer deux ou trois paramètres à ce gestionnaire.

Définition d'une valeur par défaut pour un paramètre

Pour utiliser une valeur par défaut pour un paramètre, vous vérifiez si le paramètre est vide. Si c'est le cas, aucune valeur n'a été passée et vous pouvez simplement mettre la valeur par défaut souhaitée dans le paramètre, comme dans l'exemple suivant :

```
on logData theData,theFilePath
  if theFilePath is empty then
    put "fichierjournal" into theFilePath
  end if
  put theData into URL ("file:" & theFilePath)
end logData
```

Le gestionnaire "**logData**" met les données dans un fichier, dont le nom et l'emplacement est ce que vous spécifiez dans le second paramètre. Si vous ne fournissez qu'un paramètre, le gestionnaire utilise le nom de fichier "**fichierjournal** " comme valeur par défaut, et enregistre les données de ce fichier :

```
logData field 1,"/Disk/Folder/data.txt" – spécifie un fichier
logData myVariable-- ne spécifie pas de fichier, utilise "fichierjournal"
```

La première déclaration ci-dessus indique le deuxième paramètre, donc elle n'utilise pas la valeur par défaut. La deuxième déclaration ne spécifie pas de paramètre, de sorte que les données seront placées dans le fichier "**fichierjournal**" par défaut.

5.4.6 Paramètres Implicites

Si une déclaration passe plus de paramètres que le gestionnaire de destination n'a de variables de paramètres pour les contenir, le gestionnaire récepteur peut accéder aux paramètres supplémentaires avec la fonction **param** :

```
function product firstFactor,secondFactor
  put firstFactor * secondFactor into theProduct
  if the paramCount > 2 then
    repeat with x = 3 to the paramCount
      multiply theProduct by param(x)
      end repeat
end if
  return theProduct
end product
```

La fonction ci-dessus suppose que deux paramètres à multiplier seront passés, mais peut multiplier plus de nombres en utilisant la fonction de **param** pour accéder aux paramètres au-delà des deux premiers. La déclaration suivante utilise la fonction personnalisée "product" ci-dessus pour multiplier quatre chiffres ensemble :

```
answer product(22,10,3,7)
```

Lorsque le gestionnaire «produit» s'exécute, les deux premiers paramètres - 22 et 10 - sont placés dans les

variables de paramètres "**firstFactor**" et "**secondFactor**". Le troisième paramètre, 3, est accessible avec le "**param** (3)" de l'expression, et le quatrième paramètre, 7, est accessible avec le "**param** (4)" de l'expression.

5.4.7 Passage de paramètres par référence

Normalement, si vous passez un nom de variable comme un paramètre, cette variable n'est pas modifiée par tout ce que le gestionnaire appelé, fait. C'est parce que la variable elle-même n'est pas passée, seulement son contenu. Passer des paramètres de cette façon est appelé "passage par valeur" parce que c'est la valeur de la variable - pas la variable elle-même - qui est passée.

Si le nom d'une variable de paramètre est précédé par le caractère **@**, la valeur de ce paramètre est un nom de variable, plutôt que la valeur de la variable. Changer la variable de paramètre dans le dit gestionnaire, modifie la valeur de la variable dans le gestionnaire d'appel. Cette façon de passer des paramètres est appelé "passage par référence", parce que vous passez une référence à la variable elle-même au lieu de sa valeur.
Par exemple, le gestionnaire suivant prend un paramètre et lui ajoute 1 :

```
on setVariable @incomingVar
  add 1 to incomingVar
end setVariable
```

Le gestionnaire suivant appelle le gestionnaire "setVariable" ci-dessus :

```
on mouseUp
   put 8 into someVariable
   setVariable someVariable -- call by reference
   answer "someVariable is now:" && someVariable
end mouseUp
```

L'exécution de ce gestionnaire **mouseUp** affiche une boîte de dialogue qui dit "**someVariable** est maintenant : 9". Puisque "**someVariable**" a été passé par référence au gestionnaire **setVariable**, sa valeur a changé quand **setVariable** ajoute 1 à la variable du paramètre correspondant.

Vous pouvez passer des paramètres par référence à une fonction ou une commande personnalisée, simplement en faisant précéder le nom du paramètre avec le caractère **@** dans la première ligne du gestionnaire, comme dans le gestionnaire **setVariable** de l'exemple ci-dessus. Ne pas utiliser le caractère **@** pour désigner le paramètre ailleurs dans le gestionnaire.

Note: Si un paramètre est passé par référence, vous ne pouvez passer que les noms de variable pour ce paramètre. Vous ne pouvez pas passer, les chaînes littérales ou des expressions, à l'aide d'autres conteneurs comme les champs. Tenter d'utiliser la commande **setVariable** comme décrit ci-dessus en utilisant les paramètres suivants provoquera une erreur d'exécution **:**

```
setVariable 2 -- ne peut pas passer une référence littérale
setVariable field 2 -- ne peut pas passer un conteneur
setVariable line 1 of someVariable -- ne peut pas passer un bloc
```

Paramètres Vides
Si un gestionnaire définit un paramètre comme étant passé par référence, vous devez inclure ce paramètre lors de l'appel du gestionnaire. Omettre cela provoquera une erreur d'exécution.

5.4.8 Retourner les Valeurs

Une fois qu'un gestionnaire de fonction a calculé une valeur, il y a besoin d'un moyen de renvoyer le résultat au gestionnaire qui a appelé la fonction. Et si une erreur se produit lors d'un traitement de messages, il y a besoin d'un moyen d'envoyer un message d'erreur au gestionnaire d'appel.
La structure de contrôle de retour est utilisée dans un gestionnaire de fonction pour passer la valeur résultante vers le gestionnaire d'appel. La valeur retournée est substituée à l'appel de fonction dans l'instruction d'appel, tout comme la valeur d'une fonction intégrée. Un autre regard sur l'exemple ci-dessus :

```
function expanded theString
  repeat for each character nextChar in theString
    put nextChar & space after expandedString
  end repeat
  return char 1 to -2 of expandedString
end expanded
```

Dans l'exemple de fonction personnalisée ci-dessus, la structure de contrôle **return** renvoie la chaîne traitée, en retour au gestionnaire mouseUp qui a appelé la fonction "**expended**".

Note : La structure de contrôle **return** arrête le gestionnaire, il est donc généralement la dernière ligne dans le gestionnaire**.**

5.4.9 Retourner une erreur à partir d'un gestionnaire de message

Lorsqu'elle est utilisée dans un gestionnaire de messages, la structure de contrôle **return** sert un but légèrement différent : elle renvoie un message d'erreur au gestionnaire d'appel.
Lorsqu'elle est utilisée dans un gestionnaire de messages, la structure de contrôle **return** définit la fonction de résultat pour le gestionnaire d'appel.
Si vous voulez retourner un message d'erreur au gestionnaire d'appel, utiliser la structure de contrôle **return** dans le gestionnaire de message.

Voici un exemple d'un gestionnaire de message qui affiche une boîte de dialogue :

```
on echoAMessage
  ask "Que voulez vous montrer ?"
  if it is empty then return "Pas de message !"
  else answer it
end echoAMessage
```

Ce gestionnaire demande à l'utilisateur d'entrer un message, puis affiche ce message dans une boîte de dialogue. Si l'utilisateur n'entre rien (ou clique sur Annuler), le gestionnaire envoie un message d'erreur au gestionnaire d'appel. Un gestionnaire qui utilise la commande personnalisée "**echoAMessage**" peut vérifier le résultat de la fonction pour voir si la commande affiche avec succès un message :

```
on mouseUp
  echoAMessage
  if the result is empty then beep
end mouseUp
```

Note : La fonction **result** est également fixée par plusieurs commandes intégrées dans le cas d'une erreur. Si vous vérifiez **result** dans un gestionnaire, la valeur appartient à toute commande - intégrée ou personnalisée - qui l'a établie en dernier, donc si vous allez vérifier **result**, assurez-vous de le faire juste après la commande dont vous voulez pour vérifier le succès.

5.4.10 Résumé

Dans cette rubrique, vous avez appris que:

- Une commande indique à l'application de faire quelque chose, alors qu'une fonction demande à l'application de calculer une valeur.
- Vous pouvez créer une commande personnalisée ou une fonction personnalisée en écrivant un gestionnaire pour cela.
- Les valeurs que vous transmettez à un gestionnaire sont appelés paramètres.
- Pour passer un paramètre par référence, vous faites précéder son nom par un signe @ dans la première ligne du gestionnaire.
- Lorsqu'elle est utilisée dans un gestionnaire de fonction, la structure de contrôle **return** renvoie une valeur.
- Lorsqu'elle est utilisée dans un gestionnaire de messages, la structure de contrôle **return** renvoie un message d'erreur qui peut être consulté avec la fonction **result**.

5.5 Variables

Une variable est un lieu pour stocker les données que vous créez, qui n'a pas de représentation à l'écran. Les variables peuvent contenir toutes les données que vous souhaitez placer en elles. Une manière d'envisager une variable peut-être aussi une boîte avec un nom dessus. Vous pouvez mettre ce que vous voulez dans la boîte, et de le ressortir plus tard, en fournissant simplement le nom de la variable :

```
put 1 into cetteChose -- création d'une variable nommée "cetteChose" avec la valeur 1
put cetteChose into field ID 234
put "Salut encore !" into line 2 of cetteChose
```

Mais contrairement à d'autres types de conteneurs, les variables sont non-permanentes et ne sont pas enregistrées avec la pile. Au lieu de cela, les variables sont automatiquement supprimées, soit quand leur gestionnaire a fini de s'exécuter, soit quand vous quittez l'application (en fonction de la portée de la variable). Vous pouvez également utiliser la commande **delete variable** pour supprimer une variable. Lorsqu'une variable est supprimée, non seulement le contenu de la variable disparaît, mais aussi la variable elle-même -- la "boîte" elle même.

Astuce : Pour enregistrer la valeur d'une variable, définir une propriété personnalisée de la pile à la valeur de la variable dans vos gestionnaires **closeStackRequest** ou **shutDown** de vos applications. Pour restaurer la variable, mettre la propriété personnalisée dans une variable dans le gestionnaire **startup** ou **openStack** de l'application.

5.5.1 Variable Scope

La portée d'une variable est la partie de l'application où la variable peut être utilisée. Si vous faites référence à une variable d'un gestionnaire qui est en dehors du champ de la variable, vous obtenez soit une erreur d'exécution soit un résultat inattendu. Il existe trois niveaux de portée des variables : local, script local et global. Chaque variable est l'un de ces trois types. La différence entre ces périmètres est comment on peut les utiliser et la durée de leur valeur.

5.5.2 Variables Locales

Une variable locale ne peut être utilisée que dans le gestionnaire qui la crée. Une fois que le gestionnaire a fini de s'exécuter, la variable est supprimée. La prochaine fois que vous exécuterez le gestionnaire, la variable commencera à partir de zéro : elle ne retient pas ce que vous avez mis dedans la dernière fois que le gestionnaire a été exécuté.

Pour créer une variable locale, il vous suffit de mettre quelque chose en elle. Si vous utilisez la commande **put**

avec un nom de variable qui n'existe pas encore, la variable est automatiquement créée comme une variable locale :

```
put true into myNewVar -- crée une variable nommée "myNewVar"
```

Astuce : Alors que vous pouvez utiliser presque n'importe quel mot qui n'est pas un mot du langage de LiveCode - dit mot réservé - pour un nom de variable, cela vous aidera grandement si vous prenez l'habitude de nommer des variables logiquement et systématiquement.

Pour plus de détails sur les noms autorisés pour les variables, voir la section sur les noms de variables ci-dessous. Pour des recommandations sur l'appellation des variables, voir la section sur les Conseils pour l'écriture du Bon Code ci-dessous.

Sinon, vous pouvez créer une variable locale en la déclarant explicitement à l'aide de la commande **local** à l'intérieur d'un gestionnaire:

```
local myNewVar -- crée une variable nommée "myNewVar"
put true into myNewVar -- place une valeur dans "myNewVar".
```

Important : Si vous utilisez une variable locale dans un gestionnaire, et ce gestionnaire appelle un autre gestionnaire, vous ne pouvez pas utiliser la variable locale dans la deuxième gestionnaire.

Si vous utilisez une variable avec le même nom, LiveCode crée une seconde variable qui est locale au second gestionnaire. Mais les deux variables locales n'ont pas d'incidence l'une sur l'autre, parce qu'elles sont dans différents gestionnaires.

Dans l'exemple suivant, les deux gestionnaires ont chacun une variable locale nommée «myVar», mais ce sont des variables locales différentes parce qu'elles sont dans différents gestionnaires, et un changement dans l'une n'affecte pas l'autre :

```
on mouseUp
  put 1 into myVar -- création de la variable locale myVar
  doCalledHandler
  answer myVar -- affiche "1", et non "2"
end mouseUp

on doCalledHandler
  put 2 into myVar -- création d'un variable différente avec le même non
end doCalledHandler
```

Une source commune de bugs implique une faute d'orthographe d'un nom de variable locale. Normalement, ceci ne produit pas d'erreur d'exécution, parce que si vous utilisez une variable qui n'existe pas, LiveCode la crée automatiquement. Cela signifie que si vous avez mal orthographié un nom de variable, LiveCode crée une nouvelle variable avec le nom mal orthographié. Ce bug peut être difficile à traquer car il peut entraîner une variable ayant une valeur erronée sans provoquer un message d'erreur.
Pour éviter ce problème, vous pouvez exiger que toutes les variables locales soit déclarées avec la commande **local**. Vous faites cela en activant **Script** / **Variable Checking** dans barre de menu de l'éditeur de code.
Si cette option est activée, essayer d'utiliser une variable locale qui n'existe pas provoquera une erreur d'exécution, au lieu de la créer automatiquement.
Tous les noms de variables mal orthographiés vont donc provoquer une erreur d'exécution évidente lorsque leur gestionnaire est exécuté, ce qui les rend faciles à trouver. Les variables locales sont supprimées lorsque le gestionnaire qui les utilise arrive en fin d'exécution. Vous pouvez également utiliser la commande **delete variable** pour supprimer une variable locale.

5.5.3 Variable Locale de Script

Une variable locale de script peut être utilisée par n'importe quel gestionnaire dans le script d'un objet. Vous ne pouvez pas utiliser une variable locale de script dans les gestionnaires des scripts des autres objets.

Contrairement à une variable locale, une variable locale de script conserve sa valeur même après qu'un gestionnaire ait fini de s'exécuter.

Pour créer une variable locale de script, vous devez utiliser la commande **local** dans le script, mais en dehors de tout gestionnaire.

Nous vous recommandons de toujours déclarer des variables locales de script dans la partie supérieure d'un script afin qu'elles soient toujours au même endroit et faciles à trouver :

```
local mySharedVariable

on mouseDown
  put 2 into mySharedVariable
end mouseDown

on mouseUp
   answer mySharedVariable -- displays "2"
end mouseUp
```

Vous pouvez également utiliser la commande **delete variable** pour supprimer une variable locale de script.

Note : Si vous mettez la commande **local** dans un gestionnaire, et non en dehors de tout gestionnaire, cela crée à la place une variable locale. La commande ne crée une variable locale de script que si vous la mettez a la racine du script, non dans un gestionnaire.

5.5.4 Variables Globales

Une variable globale peut être utilisée par n'importe quel gestionnaire, n'importe où dans l'application. Contrairement à une variable locale, une variable globale conserve sa valeur même après que le gestionnaire l'ayant créé ait fini de s'exécuter. Contrairement à une variable locale de script, une variable globale peut être utilisée par n'importe quel gestionnaire dans le script de n'importe quel objet. La même variable globale peut être utilisée par n'importe quelle pile pendant une session. Vous pouvez déclarer une variable globale dans une pile, et l'utiliser dans d'autres.
Pour créer une variable globale, vous devez la déclarer en utilisant la commande **global** :

```
global someGlobalSetting
```

Important : Vous devez également utiliser la commande **global** au début d'un gestionnaire pour rendre une variable globale déjà existante, disponible à ce gestionnaire. Alors qu'une variable globale peut être utilisée par n'importe quel gestionnaire, vous devez faire ceci pour n'importe quel gestionnaire qui devra utiliser cette variable globale. Si vous ne déclarez pas une variable globale avant de l'utiliser, le gestionnaire n'aura pas conscience de l'existence de la variable globale attendue, et il se contentera de créer une variable locale du même nom.

Vous pouvez utiliser la commande **global** soit à l'intérieur d'un gestionnaire, soit en dehors de tout gestionnaire au sommet d'un script (comme un script local).

Si vous utilisez la commande dans un gestionnaire, la variable globale peut être utilisée par toute déclaration

dans ce gestionnaire.
Si vous utilisez la commande dans un script, mais en dehors de tout gestionnaire, la variable globale est disponible pour chaque gestionnaire de ce script.

L'exemple suivant illustre l'utilisation d'une variable globale dans un script de bouton.

Dans cet exemple, la variable est déclarée en dehors de tout gestionnaire, de sorte que les gestionnaires individuels n'ont pas besoin de déclarer à nouveau :

```
global myGlobal -- déclaration de la variable globale pour tout le script

on mouseDown -- peut utiliser "myGlobal"
  put 1 into myGlobal
end mouseDown

on mouseUp -- peut utiliser "myGlobal"
  add 2 to myGlobal
  answer myGlobal -- affiche "3"
end mouseUp
```

Pour utiliser la même variable globale dans un gestionnaire où la variable n'est pas déclarée dans le script, vous devez placer la déclaration globale dans le gestionnaire :

```
on mouseUp -- dans le script d'un autre bouton
  global myGlobal
  add 5 to myGlobal
  answer myGlobal
end mouseUp
```

Si vous cliquez sur le premier bouton, puis le second, le second bouton affiche le nombre 8.
Comme pour les variables locales de script, nous vous conseillons de placer toutes les déclarations global dans les scripts au début du script, de rendre les déclarations faciles à retrouver pour vous et les autres.

Astuce : Vous pouvez obtenir une liste des variables globales existantes avec la fonction GlobalNames. Vous pouvez également choisir **Development** / **Variable Watcher** pour afficher la liste des variables globales et modifier leurs valeurs. Ou vous pouvez obtenir la valeur en utilisant la boîte de message.

Les variables globales sont automatiquement supprimées lorsque vous quittez l'application. Vous pouvez également utiliser la commande **delete variable** pour supprimer une variable globale.

5.5.5 Nommer les Variables

Les noms de variables doivent être constitués d'un seul mot et peuvent contenir n'importe quelle combinaison de lettres, chiffres et tirets bas (_). Le premier caractère doit être une lettre ou un trait de soulignement. Vous ne pouvez pas utiliser n'importe quel mot du langage LiveCode comme le nom d'une variable.

Voici quelques exemples de noms de variables possibles : uneVariable ou image3 ou mon_nouveau_fichier_sortie
Voici quelques noms qui ne peuvent pas être utilisés comme noms de variables :

3rdRock **-- illégal : commence par un nombre**
this&That **-- illégal car "&" ne peut être utilisé**
My Variable **-- illégal car plus d'un mot utilisé (espace)**

Évitez de donner à une variable le même nom qu'une propriété personnalisée. Si vous vous référez à une propriété personnalisée, et si il ya une variable du même nom, LiveCode utilise le contenu de la variable comme le nom de la propriété personnalisée. En général, cela va produire des résultats inattendus.

Attention : Les variables globales dont le nom commence par "gRev" sont réservées par l'environnement de développement LiveCode.

Voir la section Conseils pour l'écriture d'un bon code ci-dessous pour obtenir des conseils sur le choix des noms de variables.

5.5.6 Types spéciaux de variables

La plupart du temps, vous utilisez des variables que vous créez vous-même, en utilisant les commandes **local** ou **global**, ou tout simplement en mettant une valeur dans une nouvelle variable pour la créer. LiveCode crée également certains types de variables automatiquement : des variables **parameter**, des variables **command-line**, des variables **environment** et la variable spéciale **it**.

Variables de Paramètres

Dans un gestionnaire, pour une commande personnalisée ou une fonction personnalisée, vous pouvez définir les paramètres sur la première ligne du gestionnaire. Par exemple, le gestionnaire suivant définit deux paramètres nommés "**thisThing**" et "**thatThing**":

```
on myHandler thisThing,thatThing
  add thisThing to thatThing
  subtract thatThing from field 2
end myHandler
```

Lorsque vous utilisez la commande personnalisée ou une fonction personnalisée, vous pouvez passer des valeurs à l'aide de paramètres :

```
myHandler 15,4+1 -- place "15" dans le paramètre "thisThing", et met "5" dans le paramètre "thatThing"
```

Lorsque les paramètres nommés sont utilisés dans un gestionnaire, ils sont appelés variables de paramètres. Dans le gestionnaire, les paramètres peuvent être utilisés de la même manière que les variables locales : vous pouvez obtenir leur valeur, les utiliser dans des expressions, et y placer des données . Comme les variables locales, les variables de paramètres subsistent seulement tant que le gestionnaire est en cours d'exécution.

Variables d'Environnement

La plupart des systèmes d'exploitation pris en charge par LiveCode fournissent des informations sur l'environnement d'exploitation avec les variables d'environnement.
Vous pouvez accéder aux variables d'environnement en faisant précéder le caractère **$** pour le nom de la variable. Par exemple, l'instruction suivante récupère le contenu de la variable d'environnement LOGNAME, qui détient le nom de connexion de l'utilisateur actuel :

```
get $LOGNAME
```

Consultez la documentation technique de votre système d'exploitation pour savoir quelles variables d'environnement sont disponibles.

Vous pouvez également créer vos propres variables d'environnement en faisant précéder le caractère $ pour le nom de la nouvelle variable d'environnement :

```
put field 3 into $MYENVVAR
```

Note : Les variables d'environnement se comportent comme des variables globales et peuvent être utilisées dans n'importe quel gestionnaire. Cependant, vous n'avez pas besoin d'utiliser la commande **global** pour les déclarer avant de les utiliser.

Les variables d'environnement que vous créez de cette façon sont disponibles à la demande, et sont également exportées vers les processus lancés par la fonction **shell** ou la commande d'ouverture du processus.

Variables arguments ligne de commande

Si vous démarrez l'application à partir d'une ligne de commande, le nom de la commande est stockée dans la variable **$0** et tout argument transmis à la ligne de commande sera stocké dans des variables numérotées en commençant par le caractère **$**.
Par exemple, si vous démarrez l'application en tapant la commande **shell** suivante :

```
myrevapp -h name
```

alors la variable **$0** contient **myrevapp** (le nom de l'application), **$1** contient **-h**, et **$2** contient **name**.

Note: Les variables des arguments de ligne de commande se comportent comme des variables globales et peuvent être utilisées dans n'importe quel gestionnaire. Cependant, vous n'avez pas besoin d'utiliser le commande **global** pour les déclarer avant de les utiliser.

La variable spéciale "it"

La variable **it** est une variable locale spéciale utilisée par LiveCode pour stocker certains résultats.

Certaines commandes - comme **get**, **convert**, **read from file**, **ask** et **answer** - mettent leurs résultats dans cette variable spéciale.

Pour une liste complète des commandes qui utilisent cette variable, reportez-vous à l'entrée sur **it** dans le dictionnaire de LiveCode.

L'exemple suivant montre comment la commande **answer** utilise la variable **it** :

```
on mouseUp
  answer "Où aller ?" with "Reculer" or "Avancer"
  -- la commande answer récupère le bouton cliqué par l'utilisateur
  -- dans la variable it :
  if it is "Reculer" then go back
  else go next
end mouseUp
```

Vous pouvez utiliser la variable **it** de la même manière que toute autre variable locale, en utilisant la commande **put** pour y placer les valeurs et en l'utilisant dans les expressions nécessitant de travailler avec son contenu.

5.5.7 Variables Array (tables)

Une variable peut détenir plus d'une seule valeur. Une variable qui contient plus d'une valeur est appelée unetable (**array**), et chacune des valeurs qu'elle détient est appelé un élément. Chaque élément a son propre nom (appelé la clé de l'élément).
Si vous imaginez une variable comme une boîte avec un nom, vous pouvez imaginer une matrice comme une boîte avec des compartiments à l'intérieur. Chaque compartiment est un élément, et chaque compartiment a un nom, ou la clé, qui lui est propre.
Vous spécifiez un élément d'une variable **array** en utilisant le nom de la variable avec la clé de l'élément. Vous placez la clé entre crochets. La clé peut être un nom, un numéro ou une variable. Voici un exemple qui montre comment mettre des données dans un seul élément d'une table:

```
put "ABC" into maVariable["monNomCle"]
```

Note : Si vous utilisez une clé qui n'est pas un nombre ou une variable, vous devez placer le nom de la clé entre guillemets chaque fois que vous y faites référence. Cela évite les problèmes au cas où il y ait une variable, ou un mot réservé, avec le même nom.

Vous pouvez utiliser n'importe quel élément d'une variable **array** de la même façon que vous utilisez toute variable : mettre les données dans l'élément (ou avant ou après celle-ci) et de savoir quelles sont les données qu'il contient. De cette façon, n'importe quel élément d'une variable **array** est comme une variable en elle-même.
Les éléments d'une variable **array** peuvent contenir des éléments imbriqués ou sous-éléments, la rendant multi-dimensionnelle. Ce type de tableau est idéal pour le traitement des structures de données hiérarchiques comme les arbres ou **XML**. Pour accéder à un sous-élément, il suffit de le déclarer en utilisant un ensemble supplémentaire de crochets.

```
put "ABC" into maVariable["monNomCle"][“uneDonneeFille”]
```

Vous pouvez imbriquer des éléments jusqu'à n'importe quel nombre de niveaux.

Effacer des Eléments d'une Variable Array

Vous utilisez la commande **delete variable** pour supprimer un élément d'une variable de **matrice**, de la même manière que vous supprimez une variable classique. Pour supprimer un élément, vous spécifiez la variable et la clé de l'élément :

```
delete variable maVariable["monElement"]
delete variable maVariable["monElement"] ["elementEnfant1"]
```

Cette déclaration supprime l'élément "monElement" de la variable **maVariable**, mais n'efface pas les autres éléments de la variable **array**.

Astuce : Pour supprimer le contenu d'un élément sans supprimer l'élément lui-même, mettre **empty** dans l'élément.

```
put empty into myVar["myElement"]
```

Lister les Eléments dans une Variable Array

Vous utilisez la fonction **keys** pour dresser la liste des éléments dans une variable **array**. La fonction **keys** retourne une liste d'éléments, un par ligne :

```
put the keys of myArray into listOfElements
```

Liste des éléments imbriqués dans un élément

Vous utilisez la fonction **keys** pour dresser la liste des éléments enfants d'un élément dans une variable **array**. La fonction **keys** renvoie la liste d'éléments, un par ligne :

```
put the keys of myArray[“node25”] into listOfElements
```

Transformer une liste de données en variable Array

La commande **split** sépare les parties d'une variable en éléments de variable **array**. Vous pouvez spécifier par quel(s) caractère(s) dans la liste vous souhaitez que les données soient partagées. Les données seront converties en un ensemble d'éléments nommés selon l'endroit de la division. Par exemple :

```
put "A pomme,B bouteille,C panier" into maVariable
split maVariable by comma and space
```

Se traduira par ce qui suit :

KEY	VALUE
A	pomme
B	bouteille
C	panier

Pour plus de détails, voir la commande **split** dans le Dictionnaire LiveCode.

Combiner les éléments d'untableau dans une liste

La commande combine regroupe les éléments du tableau en une seule variable. Après que la commande ait terminé son exécution, la variable n'est plus de type **array**.

Par example:

```
combine monArray using return
```

Cette commande va combiner le contenu de la chaque élément de la variable **array** d'origine afin qu'ils apparaissent sur une ligne distincte.

Pour plus d'informations, voir la commande **combine** dans le Dictionnaire LiveCode.

Imbriquer un Array

Vous pouvez placer une variable array entiere comme un enfant d'un élément en mettant une variable array dans un élément d'une autre variable array. Par exemple :

```
put tMonArray into tGrosArray[“node50”]
```

Se traduira par l'ensemble de la variable tMonArray copié comme enfant de node50 au sein tGrosArray.

Plus d'Information

Pour plus d'informations sur l'utilisation de variables array, voir la section sur le Traitement Texte et Données.

5.5.8 Constantes

Une constante est une valeur qui a un nom. Comme une variable, une constante est définie dans votre script. Contrairement aux variables, les constantes ne peuvent pas être modifiées. Lorsque vous utilisez une constante, LiveCode, substitue la valeur de la constante à son nom. L'exemple suivant utilise une constante nommée **slash** :

```
put slash after field "Expressions" -- displays "/"
```

Vous créez une nouvelle constante en utilisant la commande constant. Vous ne pouvez pas mettre n'importe quoi dans une constante une fois qu'elle a été créée.

Constantes intégrées

La language LiveCode définit plusieurs constantes, telles que le **return**, **space** et **comma** (virgule), pour les caractères ayant une signification particulière dans les scripts et qui par conséquent ne peuvent pas être entrés littéralement dans une expression.

Astuce : Pour voir une liste de toutes les constantes intégrées, ouvrez la fenêtre de documentation, cliquez Dictionnaire LiveCode, et choisissez **constant** dans le menu en haut de la fenêtre.

Constantes définies par l'utilisateur
Vous pouvez aussi définir vos propres constantes à l'aide de la commande **constant** :

```
constant monNom="Gédéon Dulation"
```

Comme les variables, des constantes peuvent avoir une portée différente selon la façon dont elles sont créées. Une constante peut être définie comme une constante locale ou une constante locale de script :
Si vous placez la commande **constant** dans un gestionnaire, la constante peut être utilisée uniquement dans ce gestionnaire. Si vous placez la commande **constant** dans un script, mais en dehors de tout gestionnaire, la constante peut être utilisée dans n'importe quel gestionnaire du script.

5.6 Conteneurs, Opérateurs et Sources de Valeur

5.6.1 Qu'est-ce qu'un conteneur ?

Les conteneurs sont des sources d'information qui peuvent être modifiées en utilisant des expressions **chunk** (fractions, parties). En plus des variables, LiveCode a six autres types de conteneurs: champs, boutons, images, **URL**, boîte de sélection, et boîte de message. Les champs, boutons, et des images importées sont tous des objets LiveCode. Tous affichent leur contenu sur l'écran, de différentes manières, et le contenu de tous les trois sont sauvegardés lorsque vous enregistrez la pile. les **URL** renvoient à des ressources externes (fichiers sur le système, ou des articles sur un serveur Internet). La boîte de message est un conteneur spécial qui fait partie de l'environnement de développement. Toutes ces conteneurs sont traités plus en détail dans leurs sections respectives.

5.6.2 Configurer et récupérer des données à partir de conteneurs

Vous utilisez la commande **put** pour placer les données dans un conteneur, ou pour placer les données à partir d'un conteneur dans un autre conteneur ou variable. Les conteneurs supportent l'utilisation

d'expressions **chunk**, donnant la capacité de préciser une partie (bloc) de leur contenu en se référant à lui en anglais. Pour plus de détails, voir la section *Expressions* ***Chunk***.

5.6.3 Quelles sont les sources de valeur ?

Les sources de valeur sont comme des conteneurs. Elles peuvent être récupérées en utilisant la commande **get**. Toutefois, contrairement aux conteneurs, elles ne peuvent pas être définies à l'aide de la commande **put**. Les sources de valeur incluent les propriétés, les appels de fonction, les chaînes littérales et les constantes.

5.6.4 Obtention et définition des propriétés

Vous pouvez utiliser la commande **get** pour récupérer des données de propriétés. Les propriétés peuvent être définies à l'aide de la commande **set**. Lors de la récupération de données, vous pouvez utiliser des expressions de **chunk**. Cependant, vous devez définir une propriété dans sa totalité, en utilisant la commande **set**. Les propriétés sont abordées plus en détail dans la section sur Propriétés. Des exemples de récupération et de définition des propriétés se trouvent tout au long de ce guide de l'utilisateur.

5.6.5 Les Chaînes Littérales

Une chaîne littérale est une chaîne de caractères dont la valeur est elle-même. Si la chaîne est un nombre, la valeur correspond à ce nombre.

5.6.6 Utiliser des Chaînes Littérales

Lorsque vous utilisez une chaîne littérale dans une expression, LiveCode remplace simplement la chaîne par elle-même :

```
put "Hello World!" into field 1
get 1 + 2 + it
put 1 - 4.234 into field "Result"
```

Chaines entre guillemets

Les chaînes littérales qui se composent de plus d'un mot ou des mots réservés du language LiveCode doivent être placés entre guillemets :

```
put "This is a test" into myVar -- OK
put This is a test into myVar  --Ne fonctionne pas – guillemets absents
put That into myVar -- Fonctionne
put This into myVar – Ne fonctionne pas – mot réservé
```

Dans certains contextes, vous pouvez utiliser une chaîne littérale d'un mot sans guillemets sans provoquer une erreur de script. Toutefois, vous devriez prendre l'habitude de toujours mettre entre guillemets les chaînes littérales (autre que des chiffres), car cela garantit que la déclaration va continuer à fonctionner correctement même si la chaîne devient un mot réservé LiveCode à l'avenir.

Si l'option **Script** / **Variable Checking** est définie sur vrai, la compilation d'un script qui contient une chaîne littérale sans guillemets provoque une erreur de script.

5.6.7 Les Opérateurs

Utilisez les opérateurs pour mettre ensemble, pour comparer ou pour effectuer une opération sur les données. Utiliser un opérateur de chaîne pour combiner des données. Utilisez un opérateur numérique pour effectuer un calcul. Enfin, utilisez un opérateur logique pour retourner un vrai ou faux.

5.6.8 Opérateurs Numériques

Les opérateurs numériques génèrent un nombre comme résultat. Les opérateurs numériques comprennent les opérateurs arithmétiques (**+**, **-**, *****, **/**, **mod**, **div**, et le **^**) et les opérateurs au niveau du bit (**bitAnd**, **bitOr**, **bitXOr** et **bitNot**).
Pour obtenir des instructions d'utilisations individuelles, consulter l'opérateur que vous souhaitez utiliser dans le Dictionnaire LiveCode. Par exemple :

```
put "1+2 =" && 1+2 into field "Eq" -- affiche "1+2 = 3"
```

Opérateurs de chaînes

Les opérateurs de chaînes produisent une chaîne de caractères comme résultat. Les opérateurs de chaîne sont les opérateurs de concaténation (**&**, **&&**, et **,**).

```
put "1+2 =" && 1+2 into field "Eq" -- affiche "1+2 = 3"
```

1 seul **&** concatène 2 éléments, 2 **&** (**&&**) concatènent et ajoutent un espace entre les deux éléments

5.6.9 Opérateurs Logiques

Les opérateurs logiques produisent soit "vrai" ou "faux" comme résultat.

Les opérateurs logiques incluent les opérateurs de comparaison (**=**, **<>**, **<**, **>**, **<=**, **>=**), les opérateurs d'existence (**there is a** (il ya), **there is no** (il n'existe pas), **is in** (est dans), **is not in** (n'est pas dans), **is among** (est parmi), **is not among** (ne fait pas partie), **contains** (contient)) , les opérateurs de type de données (**is a** (est), **is not a** (n'est pas un)), les opérateurs de géométrie (**is within** (est à l'intérieur), **is not within** (n'est pas à l'intérieur)) et les opérateurs logiques de base (**and** (et), **or** (ou), **not** (non)).

```
if the platform is "MacOS" and field "Time" < zero then...
```

5.6.10 Les opérateurs binaires versus unaire

Les opérateurs peuvent utiliser soit un seul argument (un opérateur unaire) ou deux arguments (un opérateur binaire):

```
"a" & "b" -- & est un opérateur binaire
there is a field "c" -- there est un opérateur unaire
```

Les opérateurs **bitNot**, **there is a**, **there is no**, **is a**, **is not a**, et **not** sont des opérateurs unaires. Tous les autres opérateurs sont des opérateurs binaires.

5.6.11 Conversion des valeurs

LiveCode convertit les valeurs dans les expressions pour n'importe quel type de données nécessaire pour l'opération. Cette conversion se fait automatiquement, donc vous n'avez pas besoin de savoir à quel type de données que vous avez affaire à l'avance (ce qui dans d'autres langages est appelé **type casting**).

Par exemple, supposons que vous ayez fait la déclaration suivante :

```
put char 2 of "123" + char 3 of "456" into field 3
```

Le caractère 2 de la chaîne littérale "123" est la seule chaîne de caractères «2», et le caractère 3 de la chaîne "456" est la seule chaîne de caractères "6". Quand LiveCode utilise l'opérateur **+**, il convertit automatiquement ces chaînes en nombres afin qu'ils puissent être additionnés. Ensuite, il convertit le nombre résultant en une chaîne de telle sorte qu'elle peut être placée dans un champ sous forme de texte.

5.6.12 Priorité des Opérateurs

Lorsque vous combinez les sources de valeur à l'aide d'opérateurs, LiveCode évalue chaque source de valeur dans l'expression.
Ensuite, il applique tous les opérateurs pour arriver à une valeur finale de l'expression. LiveCode n'applique pas nécessairement tous les opérateurs dans l'ordre de droite à gauche.
Au lieu de cela, il utilise l'ordre de priorité des opérateurs pour déterminer comment calculer les expressions qui incluent plus d'un opérateur. La priorité détermine l'ordre dans lequel LiveCode effectue les calculs dans les expressions. Si une expression contient plusieurs opérateurs, les opérateurs disposant d'une plus grande priorité sont calculés avant opérateurs de priorité inférieure.

Priorité des Opérateurs

Grouper	**()**	Toute partie de l'expression entre parenthèses est évaluée en premier. Si les parenthèses sont imbriquées, les valeurs les plus intimes sont évaluées en premier.
Unaire	**-** **not** **bitNot** **there is a** **there is no**	Ensuite, les opérations unaires (qui agissent sur un seul non opérande) sont effectuées. Cela inclut une soustraction unaire (qui produit un nombre négatif).
Elévation à la puissance	**^**	Ensuite, les opérations d'élévation à une puissance sont faites.
Multiplication Division	***** **/** **div** **mod**	Suivent la multiplication et la division. Ce sont des opérateurs numériques et le résultat est un nombre.
Addition	**+** **-**	Suivent l'addition et la soustraction. Ce sont des opérateurs numériques et le résultat est un nombre.
Concaténation	**&** **&&** **,**	Suivent les opérations qui joignent deux chaines. Ce sont des opération sur des chaines de caractères et le résultat est une chaine de caractère.
Comparaison	**<, <=, ≤** **>, >=, ≥** **contains** **is among** **is not among** **is in** **is not in** **is within** **is not within** **is a** **is not a**	Suivent les opérations qui comparent deux valeurs. Ce sont des opérateurs logiques et le résultat est soit vrai soit faux.

Egalité	**=, is** **<>, !=, ≠, is not**	Suivent les opérations qui comparent deux valeurs en terme d'égalité. Ce sont des opérateurs logiques et le résultat est soit vrai soit faux.
bitAnd	**bitAnd**	Suivent les opérations **bitAnd** qui permet d'opérer directement sur les bits des deux numéros. Effectue une opération "bit à bit et" sur la représentation binaire de deux nombres.
bitXOr	**bitXOr**	Suivent les opération **bitXOr** qui permet d'opérer directement sur les bits des deux numéros. Effectue une opération "bit à bit et sur la représentation binaire de deux nombres.
bitOr	**bitOr**	Suivent les opération **bitOr** qui permet d'opérer directement sur les bits des deux numéros. Effectue une opération "bit à bit et" sur la représentation binaire de deux nombres.
and	**and**	Suit l'opérateur **and**, pour combiner deux ou plusieurs valeurs logiques. Prend la valeur vrai si les deux opérandes sont vrai, faux, autrement.
or	**or**	Pour finir, l'opérateur **or** pour combiner deux ou plusieurs valeurs logiques (vraies ou fausses).
Appels de fonction	Les fonctions sont évaluées après que tous les opérateurs possibles dans les paramètres de la fonction aient été évalués.	

5.6.13 Utilisation de l'opérateur de groupement *( )*

Supposons que vous vouliez changer la priorité utilisée dans une expression. L'opérateur de regroupement **( )** a une priorité plus élevée que tous les autres opérateurs, de sorte que vous pouvez l'utiliser pour modifier l'ordre dans lequel les opérateurs sont évalués. En d'autres termes, si vous placez une partie d'une expression entre parenthèses, toutes les opérations à l'intérieur des parenthèses sont évaluées avant toute autre.

Par exemple, les appels de fonction ont une priorité plus élevée que la division. L'expression du sinus de 1/4 signifie "obtenir le sinus de 1, puis le diviser par 4" :

```
get the sin of 1/4 -- calcule la fonction sinus en premier
```

Si vous voulez le sinus de 1/4, vous devez modifier l'ordre d'évaluation de sorte que la division est faite en premier. Vous faites cela en entourant la partie que vous voulez évaluer en premier avec des parenthèses :

```
get the sin of (1/4) -- calcule la division en premier
```

Si les parenthèses sont imbriquées, l'expression dans l'ensemble le plus profond entre parenthèses est évaluée en premier.

5.6.14 Facteurs and Expressions

Une expression est une source de valeur, ou une combinaison de sources de valeur. L'ensemble des sources de valeur présentées ci-dessus - les conteneurs, les propriétés, les appels de fonction, les chaînes littérales et les constantes - sont des expressions simples. Vous utilisez des opérateurs pour combiner des sources de valeur pour produire des expressions plus complexes.

5.6.15 Définition des facteurs

Un facteur est la première partie entièrement résolue d'une expression. (Tous les facteurs sont des expressions, mais toutes les expressions ne sont pas des facteurs.)

Un facteur peut être une source de valeur ou une expression qui combine des sources de valeur, mais ne n'inclue pas les opérateurs binaires en dehors des parenthèses.

Enfermer une expression entre parenthèses la transforme en facteur.

Ces exemples montrent certaines expressions, ainsi que le premier facteur dans chaque expression, pour vous aider à voir la distinction :

Expression :	**Son premier facteur :**
3+4	3
(3+4)	(3+4)
(3 + 4)/field 4	(3+4)
field 6*pi	field 6
sin of pi/4	sin of pi
sin(pi/4)	sin(pi/4)
sin(pi/4) * exp2(12)	sin(pi/4)
whole world	whole
"whole world"	"whole world"

La distinction entre les facteurs et les expressions est en question lorsque vous utilisez la forme **the** des fonctions intégrées, lorsque vous utilisez **URL**, and lorsque vous faites référence à des objets.
Si vous utilisez la forme **the** d'une fonction intégrée qui possède un paramètre, le paramètre de **the** doit être un facteur, non une expression :

```
get the sqrt of 4 + 5 -- produit 7
get sqrt(4+5) -- produit 3
get the sqrt of (4 + 5) -- produit 3
```

Dans le premier exemple ci-dessus, même si nous avions l'intention d'obtenir la racine carrée de 9, nous avons fini avec la racine carrée de 4. C'est parce que l'expression 4 + 5 n'est pas un facteur (car il contient un opérateur binaire qui n'est pas à l'intérieur de parenthèses). Le premier facteur dans l'expression 4 + 5 est 4, de sorte que le premier exemple obtient la racine carrée de 4 (qui est de 2), puis ajoute 5 à ce résultat. Le deuxième exemple permet d'éviter ce problème, car il n'utilise pas la forme **the** de la fonction. Puisque le paramètre est mis entre parenthèses, vous pouvez utiliser soit un facteur ou une expression, et obtenir le résultat escompté, la racine carrée de 9.
Dans le troisième exemple, nous tournons l'expression 4 + 5 en un facteur en l'entourant de parenthèses. Puisque le paramètre est un facteur, nous pouvons obtenir un résultat correct, même en utilisant la forme **the** de la fonction.

Lorsque vous faites référence à des **URL**, l'**URL** est un facteur :

```
get URL "file:myfile.txt" – fonctionne
get URL "file:" & "myfile.txt" -- NE FONCTIONNE PAS
get URL ("file:" & "myfile.txt") -- fonctionne
```

Dans le premier exemple, l'**URL** spécifié est un facteur parce que c'est une simple chaîne sans aucun opérateur.

L'**URL** dans le second exemple n'est pas un facteur, car il inclut l'opérateur binaire et, donc la commande **get** tente d'obtenir l'**URL** "file:" - qui est inexistante - et de concaténer le contenu de cette **URL** avec la chaîne "monfichier . Txt ".
Dans le troisième exemple, nous passons l'**URL** dans un facteur en l'entourant de parenthèses, ceci fournissant le résultat escompté.
Lorsqu'on se réfère aux cartes ou arrière-plans, le nom (**name**), le numéro (**number**) ou l'**ID** de l'objet est une expression :

go card 1 + 1 – **va à la carte 2**
go card 3 of background "Data" && "Cards" **-- 1ère carte avec le groupe "Data Cards"**

Toutefois, en se référant à des contrôles (y compris les groupes) ou des piles, le nom, le numéro ou l'ID de l'objet est un facteur :

answer field 1 + 10 **-- affiche le contenu du champ 1 + 10**
answer field (1 + 10) **-- affiche le contenu du champ 11**
select button "My" && "Button" **-- NE FONCTIONNE PAS**
select button ("My" && "Button") – **fonctionne**

5.7 Prendre des décisions

Vous prenez des décisions en utilisant la structure de contrôle **if** (si)... **then** (alors) ... **else** (sinon) ou, si vous voulez choisir parmi une liste d'options, utilisez la structure de commande de **switch** (commutateur).

5.7.1 If...then...else (si... alors... sinon...)

Utilisez la structure de contrôle **if** pour exécuter une instruction ou une liste de déclarations dans certaines circonstances. Par exemple, vous voudrez peut-être que votre application puisse réagir différemment en fonction d'une action de l'utilisateur.
La structure de contrôle **if** commence toujours par le mot **if**. Il existe quatre formes de de structure de contrôle **if** :

if condition **then** *Instructions* [**else** *elseInstructions*]

Cette forme peut avoir un saut de ligne devant les mots **then** ou **else** ou les deux.

```
if condition then
  Liste_Instructions
[else
  elseListe_Instructions]
end if
```

```
if condition
then  Instructions
[else
  elseInstructionsList
end if]
```

```
if condition then
  InstructionsList
else elseInstructions
```

La condition est une expression qui renvoie vrai ou faux. **InstructionsList** et **elseInstructions** se composent d'une ou plusieurs déclarations de LiveCode, et peuvent également inclure des structures de contrôle **if**, **switch**, **try**, ou **repeat**. La déclaration **elseInstructions** se compose d'une déclaration de LiveCode unique.
Si la condition est évaluée à vrai, l'instruction ou **InstructionsList** est exécutée, si la condition est fausse, l'instruction **InstructionsList** est ignorée.
Si la structure de contrôle **if** contient une clause **else**, **elseInstructions** ou **elseInstructionsList** celle-ci est exécutée si la condition est fausse. Si une structure de contrôle **if** est imbriquée dans une autre, l'utilisation de la deuxième forme décrite ci-dessus est recommandée, car les autres formes peuvent causer des ambiguïtés pour identifier quelle clause fait partie l'autre boucle **if**. La structure de contrôle **if** est la plus appropriée lorsque vous voulez vérifier une seule condition.
Si vous avez besoin de vérifier de multiples possibilités, de faire quelque chose de différent pour chacune, utilisez une structure de commande **switch** à la place.

5.7.2 Switch (commutateur)

Utilisez la structure de commande **switch** lorsque vous souhaitez choisir parmi plusieurs valeurs possibles pour une expression et exécuter un ensemble de déclarations qui dépend de la valeur.

```
switch [switchExpression]
  case {caseValue | caseCondition}
    [statementList]
  [default
    defaultStatementList]
end switch
```

La structure de commande **switch** commence par le mot **switch**, sur une seule ligne, avec un **SwitchExpression** facultatif. La ligne de commande est suivie d'une ou de plusieurs sections de cas. Chaque section de cas commence avec le mot-clé **case**, suivie soit par **caseValue** (si un **SwitchExpression** a été inclus sur la ligne de commande) ou **caseCondition** (si aucun **SwitchExpression** n'a été inclus). Si **caseValue** est égal au **SwitchExpression**, ou **caseCondition** est évalué à vrai, LiveCode commence à exécuter les instructions de la section de cas.

Les sections de cas peuvent être suivies par une section facultative par défaut. Si aucune instruction break n'a encore été rencontrée dans la structure de commande **switch**, les déclarations contenues dans la section par défaut sont exécutées. La structure de commutateur se termine par une instruction **end switch**.

SwitchExpression est une expression quelconque. **caseValue** est une expression quelconque. Si **caseValue** a la même valeur que **SwitchExpression**, la condition correspond à cette la section de cas.

caseCondition est une expression qui renvoie vrai ou faux. Si **caseCondition** est évalué à vrai, la condition s'appliquera pour cette section de cas. Chaque **statementList** se compose d'une ou plusieurs déclarations de LiveCode, et peut également inclure des **if**, **switch**, **try**, ou **repeat** des structures de contrôle. **defaultStatementList** se compose d'une ou plusieurs déclarations LiveCode.

Le flux de contrôle dans une structure de commutation est moins compliqué qu'il n'y paraît. En général, quand LiveCode entre dans une structure de commande **switch**, il vérifie la première section de cas dont **caseValue** est égale à la **SwitchExpression**, ou dont **caseCondition** est vrai. Quand un état correspondant est trouvé, toutes les déclarations qui la suivent sont exécutées - même des déclarations dans une autre section de cas - jusqu'à ce qu'une instruction **break** soit rencontrée ou que la structure de commande **switch**, se termine.

Cela signifie que **statementList** d'une section de cas ne se finit pas avec une instruction **break**, les déclarations contenues dans tous les cas, les articles suivants (et la section par défaut) sont exécutés même si ces sections de cas n'ont pas un **caseValue** qui corresponde ou une vraie **caseCondition**.

Parfois, ce comportement est utile.

Cependant, dans la plupart des cas, vous devez placer une instruction **break** à la fin de chaque **statementList**. Cela garantit que ce seul **statementList** est exécuté et que le reste est ignoré. Cela signifie également que vous pouvez attacher plus d'une option **caseValue** ou **caseCondition** à la même **statementList**, simplement en plaçant une ligne de cas au dessus de la prochaine.

L'exemple suivant émet un **bip** si la carte actuelle est soit la dernière soit la première, et va sinon à la prochaine :

```
switch (le numéro de cette carte)
  case 1
    beep
    break
  case (le nombre de cartes)
    beep
    break
  default
    go next card
end switch
```

Il n'y a aucune limite au nombre de cas que vous pouvez inclure dans une structure de commande **switch**, mais plus de cas il y aura, plus LiveCode devra évaluer les expressions et plus lentement la structure **switch** s'exécutera.

5.8 Extension du Chemin de Message

Cette section traite de la manière de prolonger le chemin du message, soit par l'ajout de bibliothèques de code pour le chemin du message, ou en envoyant des messages directement à des objets qui ne sont pas actuellement dans le chemin du message.

5.8.1 Création d'une bibliothèque de code

Une bibliothèque est un ensemble de commandes et de fonctions, personnalisées, pour une application ou une zone, spécifique, de fonctionnalité. Vous pouvez créer une bibliothèque pour n'importe quel but voulu, et y mettre toutes les commandes et fonctions personnalisées dont vous avez besoin. Les bibliothèques sont généralement utilisées pour stocker des routines qui sont communes au sein de votre application. Vous pouvez également échanger des bibliothèques utiles avec d'autres développeurs.

Pour créer une bibliothèque de code, placez les gestionnaires que vous souhaitez utiliser dans n'importe quel objet disponible dans votre pile. Cet objet est maintenant une bibliothèque de code. Utilisez ensuite la commande **insert script** pour ajouter cet objet au chemin de message.
Pour insérer le script d'une pile dans le chemin du message, utilisez la commande : **start using**.

Typiquement, si vous exécutez une de ces commandes au démarrage de votre application tous les scripts peuvent accéder aux bibliothèques dont vous avez besoin. Les bibliothèques n'ont pas besoin d'être dans la même pile ou même, dans le fichier de pile. Vous pouvez charger une pile sur le disque, puis charger les bibliothèques à l'intérieur pour les rendre accessibles à toutes les piles en cours d'exécution.

Cela facilite la conception de votre application en modules, de partager le code avec d'autres développeurs ou la mise à jour de vos bibliothèques d'application sans modifier votre application. de De la même façon, vous pouvez concevoir votre application autonome de manière à faciliter sa mise à jour à l'aide d'un petit utilitaire de patch, c'est à dire sans avoir à réinstaller l'application entière.

5.8.2 Utiliser les backScripts

Pour rendre le script d'un objet disponible pour tout autre gestionnaire dans LiveCode :

```
insert script of card "Library" into back
```

Le script d'un objet qui a été inséré en arrière plan (**back**), comme dans l'exemple ci-dessus, est appelé **backScript**. Un tel objet est placé en dernier dans le chemin des messages de tous les objets. Il reçoit les messages après tout autre objet, et juste avant que le moteur ne le reçoive. L'objet **backScript** reste dans le chemin du message jusqu'à la fin de la session (ou jusqu'à ce que vous l'enleviez avec la commande **remove script**.)
Parce qu'un **backScript** reçoit des messages après tous les autres objets dans le chemin du message, si vous placez un gestionnaire dans le **backScript**, il obtient finalement tous les messages correspondants, quel que soit leur émetteur, sauf si un autre objet le traite en premier.

5.8.3 Utiliser frontScripts (script de premier-plan)

Vous pouvez également étendre le chemin du message en plaçant des objets sur le devant du chemin de message, avant tout autre objet. Le script d'un tel objet est appelé un **frontscript**, et vous utilisez la commande **insert script** pour placer l'objet en avant :

```
insert script of button "Interceptor" into front
```

Parce que l'objet est en premier dans chemin de message, il reçoit des messages avant même la cible du message. Par exemple, si vous cliquez sur un bouton, tout objet en amont du chemin de message reçoit le message **mouseUp** d'abord, avant le bouton. Si vous placez un gestionnaire dans un **frontscript**, il reçoit tous les messages correspondants avant que tout autre objet ne puisse le manipuler.

Utilisez un **frontscript** quand vous voulez être capable de gérer un message même si l'objet cible a un gestionnaire pour lui. Par exemple, l'environnement de développement LiveCode affiche un menu contextuel lorsque vous faites **Ctrl + Maj + clic droit** sur un objet. Il le fait avec un gestionnaire **mouseDown** dans un **frontscript**. Chaque fois que vous cliquez sur un objet, le **frontscript** reçoit le message **mouseDown** en premier, et vérifie si les clés nécessaires sont pressées. Si elles le sont, le gestionnaire affiche le menu contextuel, le message **mouseDown** est pris au piège et ne va pas plus loin. Dans le cas contraire, le gestionnaire transmet le message à l'objet suivant dans le chemin du message et qui est l'objet que vous avez cliqué.

5.8.4 Utiliser un Script de Pile avec la commande *start using*

La commande **start using** est similaire à la commande **insert script**, mais ne peut être utilisée que pour placer des piles, et non d'autres types d'objets, dans le chemin de message. La commande **start using**, insère la pile à la fin du chemin de message, après la pile de l'objet et la pile principale, mais avant les objets qui ont été insérés en arrière avec **insert script**.

5.8.5 Envoyer des Messages Directement aux Objets

Si le gestionnaire que vous souhaitez utiliser n'est pas dans le chemin de message, au lieu de l'insérer dans le chemin pour en faire une bibliothèque de code, vous pouvez utiliser la commande **send** pour envoyer directement un message à l'objet dont le script contient le gestionnaire.

Astuce : Vous pouvez utiliser cette technique pour sauter une partie du chemin de message, en envoyant le message directement à un objet plus haut dans la hiérarchie.

```
on mouseUp -- dans le script du button
  send "mouseUp" to this stack
end mouseUp
```

Par exemple, supposons un gestionnaire d'une commande personnalisée appelée "**myCommand**" dans le script d'un bouton, et que vous vouliez exécuter ce gestionnaire à partir d'un script de carte. Puisque le bouton n'est pas dans le chemin de message de la carte, vous ne pouvez pas simplement utiliser la commande dans le script, car si le message "**myCommand**" passe par le chemin de message de la carte, il ne trouvera pas le gestionnaire. La déclaration suivante envoie le message directement au bouton dont le script inclut le gestionnaire :

```
send "myCommand" to button "Stuff" of card "Stuff Card"
```

Important : Lorsque vous envoyez un message à un objet, le chemin de message pour ce message commence avec l'objet cible. Par exemple, si un script de pile contient une commande **send** qui envoie un message à un bouton, le message se déplace à travers le chemin de message du bouton et non celui de la pile.

Si vous souhaitez utiliser une fonction personnalisée dont le gestionnaire n'est pas dans le chemin de message, vous pouvez utiliser la fonction **value** pour spécifier l'objet. La fonction **value** peut être considérée comme un équivalent de la commande **send** pour des appels de fonction.

Par exemple, si la carte 1 d'une pile contient un gestionnaire de fonction appelé "maFonction" vous pouvez utiliser la fonction à partir du script de carte 3 avec la déclaration suivante :

```
get value("maFonction(1,2,3)",card 1)
```

5.8.6 La Commande *send* ou la Commande *call*

La commande **call** est similaire à la commande **send**. Comme la commande **send**, la commande **call** envoie un message à l'objet spécifié, vous permettant d'utiliser des gestionnaires qui ne sont pas dans le chemin de message.

La différence entre **send** et **call** est de savoir comment ils gèrent les références d'objet dans le gestionnaire qui est déclenché par le message envoyé. Si le message est envoyé par la commande **send**, les références d'objets dans le gestionnaire sont traités par rapport à l'objet duquel vous avez envoyé le message.

Par exemple, supposons que la carte 1 d'une pile contienne le gestionnaire suivant :

```
on showCard
  answer the number of this card
end showCard
```

Si un gestionnaire dans le script de carte 3 utilise la commande **send** pour envoyer un message "showCard" à la carte 1, la boîte de dialogue affiche "1", le numéro de la carte dont, le gestionnaire est activé.
Toutefois, si le même gestionnaire utilise la commande **call** à la place, la boîte de dialogue affiche «3», le numéro de la carte qui a utilisé la commande **call**.
En d'autres termes, les gestionnaires qui sont déclenchés par le commande **send** utilisent le contexte de l'objet dans lequel le gestionnaire est, tandis que les gestionnaires qui sont déclenchés par la commande **call** utilisent le contexte de l'objet qui a déclenché le gestionnaire.

5.8.7 L'écriture de code réutilisable en utilisant les comportements

Les comportements sont une méthode pour créer des fonctionnalités communes entre les objets sans dupliquer les scripts.

Un objet avec un jeu de comportements (**behavior set**) va agir comme si son script avait été créé pour le script **behavior** du bouton. Si plusieurs objets partagent le même comportement, chacun aura son propre ensemble de variables locales de script. Toutes les références à **me**, à **owner of me**, etc se résoudront à l'objet enfant en cours d'exécution.
Les scripts de comportement doivent être stockés dans des objets bouton. Pour définir le comportement d'un objet, reportez-vous au **long Identifiant** du bouton contenant le script de comportement.

5.9 Messages Basés sur une Temporisation

Les temporisateurs vous permettent de planifier des événements devant se produire à l'avenir. Utilisez des temporisateurs pour mettre à jour l'affichage, à intervalles réguliers, traiter des données par morceaux, jouer des animations, afficher des barres d'état, et partout où vous avez besoin de planifier des événements.
Les messages peuvent être prévus avec précision à la milliseconde et exécutés plusieurs fois par seconde pour créer une animation, ou peuvent être programmés pour arriver des heures plus tard. Lorsque vous planifiez un événement pour être déclenché ultérieurement, LiveCode continue de répondre aux événements utilisateurs normalement. Cela rend les messages basés sur la temporisation idéaux pour garder votre interface utilisateur réactive tout en faisant du traitement des données ou en mettant à jour l'affichage.

5.9.1 Délivrer un Futur Message

Pour délivrer un message après une période de temps spécifiée, utilisez la forme **in *temps*** de la commande **send**.

```
send "updateStatus" to me in 20 seconds
send "updateAnimation" to me in 33 milliseconds
```

5.9.2 Répétition d'un message programmé

Si vous voulez envoyer un message à plusieurs reprises, par exemple pour dessiner en continu des trames dans une animation, il suffit d'envoyer le même message à la fin du gestionnaire de messages. L'exemple suivant démarre une animation lorsque le bouton est cliqué, puis met à jour l'image à 30 images par seconde (toutes les 33 millisecondes).

```
on mouseUp
  updateAnimation
end mouseUp
on updateAnimation
  -- insérez ici le code pour rafraîchir votre animation
  send updateAnimation to me in 33 milliseconds
end updateAnimation
```

Résultat : le message **updateAnimation** est envoyé, il mettra à jour l'écran puis, se renvoie à lui-même toutes les 33 millisecondes. Le message va continuer à être déclenché indéfiniment. C'est pourquoi il est important de vous assurer qu'il y ait une condition dans le gestionnaire de message qui sortira lorsque la tâche est accomplie, sans envoyer le message à nouveau. Sinon, vous pouvez directement supprimer tout message pour arrêter la boucle (voir la section Annuler un **timer** message ci-dessous pour plus de détails).

5.9.3 Annuler un Message Temporisé

Lorsqu'un message de temporisation est envoyé, la fonction de résultat contient l'ID du message généré. Utilisez la commande Annuler pour annuler ce message, l'empêchant d'être délivré. Dans l'exemple suivant, une boîte de dialogue s'affiche 20 secondes après que l'utilisateur déplace la souris sur un bouton. Si la souris se déplace hors du du bouton avant que les 20 secondes se soient écoulées, le dialogue ne sera pas affiché.

```
local leTimerID
on mouseEnter
  send afficheDialog to me in 20 seconds
  put the result into leTimerID
end mouseEnter

on mouseLeave
  cancel leTimerID
end mouseLeave

on afficheDialog
  answer "La souris était sur ce bouton pendant 20 secondes."
end afficheDialog
```

Important : Assurez-vous d'avoir un moyen d'annuler les messages temporisés en attente. Typiquement, vous voudrez peut-être faire en sorte que la temporisation soit annulée lorsque la carte est fermée. Par exemple, un message de temporisation qui dessine une animation sur la carte en cours va générer une erreur de script si la carte est changée et le message est toujours envoyé, car le script ne sera plus en mesure de trouver les objets.

5.9.4 Afficher une liste des Messages Temporisés en Attente

Vous pouvez obtenir une liste de tous les messages temporisés actuellement en attente à l'aide de la fonction **pendingMessages**.

```
put the pendingMessages into tMessagesList
```

tMessagesList contient maintenant une liste de messages, un par ligne. Chaque ligne se compose de quatre éléments, séparés par des virgules :

* the message ID (identifiant du message)

* the time the message is scheduled for (moment prévu pour le message)

* the message name (le nom du message)

* the long ID property of the object that the message will be sent to (la propriété long ID de l'objet à qui le message sera envoyé)

Astuce : Vous pouvez voir une liste de tous les messages en attente dans la ***Message Box*** dans l'onglet messages - la cinquième icône . Pour plus de détails, voir la section sur la ***Message Box***.

Pour plus de détails, consultez l'entrée **pendingMessages** dans le Dictionnaire du LiveCode.

5.10 Astuces pour Ecrire un Bon Code

Cela vaut la peine de prendre le temps d'établir des conventions dans la façon dont vous écrivez du code. Il y a beaucoup d'avantages à rendre votre méthode de codage cohérente. Les variables correctement nommées sont plus facile à déboguer pendant le développement parce que vous connaitrez toujours la portée d'une variable particulière (qu'il s'agisse d'une variable globale ou qu'elle s'applique seulement au gestionnaire actuel, etc.).

Cela facilitera la compréhension du code que vous avez écrit quand vous voudrez le lire 6 mois plus tard. En faisant un usage approprié des fonctions et des bibliothèques, votre programme sera plus modulaire et donc plus facile à modifier et pour y ajouter des fonctionnalités à l'avenir.
Un guide complet pour écrire un bon code est au-delà de la portée de ce manuel.

Mais nous avons pensé qu'il serait utile de vous donner quelques trucs et astuces

Nommer le Variables

Utilisez des noms de variables cohérents pour rendre votre code plus facile à comprendre. Cela peut ne pas sembler important maintenant, mais quand vous aurez oublié comment cela fonctionne 6 mois après, cela aidera à le rendre lisible.

Cela rend également plus facile pour vous d'échanger du code avec les autres membres de la communauté LiveCode, quand vous avez besoin d'obtenir de l'aide pour quelque chose.

Caractère	Exemple	Usage
g	gVar	Variable **G**lobale
l	lVar	Variable **L**ocale (handler)
s	sVar	Variable locale **S**cript
p	pVar	**P**aramètres
k	kVar	Constantes
c	cVar	Propriétés personnalisée (**C**ustom properties)

En règle générale, utilisez une variable avec juste assez de portée et pas plus pour la tâche à accomplir. En d'autres termes, si une variable locale de gestionnaire est tout ce dont vous avez besoin pour un calcul, n'utilisez pas une variable script locale. Si une variable script locale est nécessaire, ne pas utiliser une globale. Cela rend moins probable l'introduction d'erreurs inattendues dans votre code en utilisant le même nom de variable à des fins différentes. Une variable locale de gestionnaire n'a de sens que dans ce gestionnaire de sorte que vous pouvez utiliser en toute sécurité le même nom de variable dans d'autres gestionnaires.

Utilisation des Commentaires

Commentez votre code, sans hésiter. Ne pas écrire des commentaires lorsque le sens est évident. Mais même une phrase à côté d'une routine complexe aidera à vous ou d'autres personnes à mieux comprendre plus tard.

Utiliser des Fonctions

Si vous avez écrit beaucoup de code, il faut se demander s'il peut être réécrit comme une série de fonctions au lieu d'un seul gestionnaire monolithique. Cela le rend beaucoup plus facile de comprendre le rôle de chaque section de code.
Plus il y aura des fonctions «boîtes noires», qui prennent des entrées et produisent des sorties, sans dépendances, plus il sera facile, si vous le souhaitez, de changer la façon dont cet aspect du code fonctionne.

Variables explicites

Si vous travaillez sur un code important en taille, il est préférable d'activer l'option **Strict Compilation Mode** dans le menu **Preferences / Script editor** de l'IDE Live Code. Ceci vous obligera à déclarer toutes les variables (mêmes les locales de script) avant de *compiler* votre script. De même vous serez obligé de placer vos chaines de caractères entre guillemets. Cette méthode de codage peut vous aider à retrouver rapidement les erreurs. Ce qui est plus important est vous développiez votre propre cohérence de style et puis de s'y tenir. Une fois que vous avez appliqué certaines de ces techniques pendant une courte période, elles deviendront une seconde nature.

Chapitre 6 Traitement du Texte et des Données

Live Code a une capacité de traitement du texte et des données de première classe. La possibilité unique de Live Code de diviser (**chunk**) les expressions (la possibilité de se référer au texte en utilisant des déclarations en anglais comme comme *word 3 to 5 of* ***myVariable***) combinée avec d'autres puissantes fonctionnalités comme : les expressions régulières, le traitement **XML**, les tableaux associatifs, les fonctions de codage et de décodage des données, les algorithmes de compression et de chiffrement, tout cela rend facile et simple le traitement d'un texte ou de données quelque soient leur complexité.

6.1 Utiliser les Bloc d'Expressions (Chunk)

Le bloc d'expression est la première méthode pour travailler sur du texte avec Live Code. Un ***chunk*** (bloc, morceau, fraction) est un terme anglais permettant de décrire une portion exacte de texte. Vous pouvez utiliser les blocs ***chunks*** pour récupérer une portion de texte ou pour éditer un texte. Ce sujet définit les types de blocs ***chunks*** que vous pouvez utiliser et décrit la syntaxe pour les caractériser.

6.1.1 Types de blocs (Chunks)

Les types communs de chunk sont **character**, **word**, **line** ou **item**. Un **item** peut être délimité par tout caractère que vous aurez défini. Le *chunk* **token** est utile lors de l'analyse de données de script. Voici un exemple utilisant le bloc **word** :

```
put word 1 to 3 of field "text" into myVariable
```

6.1.2 Utilisation des Blocs avec les Containeurs

Vous pouvez utiliser un morceau d'un conteneur comme vous utilisez un conteneur entier.
Par exemple, vous pouvez utiliser la commande **add** pour ajouter un numéro à une ligne d'un champ

```
add 1 to word 3 of field "Numbers"
```

Vous pouvez également utiliser des expressions **chunk** à remplacer (à l'aide de la commande put) ou à supprimer (à l'aide de la commande **delete**) toute partie d'un conteneur.

6.1.3 Utilisation de Blocs avec des Propriétés

Vous pouvez utiliser des expressions de **chunk** pour lire des parties d'une propriété (comme la propriété de **script**). Cependant, puisque vous modifiez une propriété avec la commande **set** plutôt que la commande **put**, vous ne pouvez pas utiliser une expression **chunk** pour changer une partie de la valeur d'une propriété. Au lieu de cela, mettez (**put**) la valeur de la propriété dans une variable, utilisez l'expression de **chunk** pour modifier la variable, puis définissez (**set**) la propriété du contenu de la variable. L'exemple suivant montre comment modifier la troisième ligne de la propriété de script d'un objet :

```
put the script of me into tempScript
put "-- Last changed by Jane" into line 3 of tempScript
set the script of me to tempScript
```

6.1.4 Le Bloc Character

Un bloc **character** est un caractère unique, qui peut être une lettre, un chiffre, un signe de ponctuation ou un caractère de contrôle.
Un **character** ne peut pas contenir tout autre type de bloc. Il peut être contenu dans tout autre type de bloc. Vous pouvez utiliser l'abréviation **char** comme un synonyme de **character** dans une expression **chunk**.

6.1.5 Le Bloc Word

Le bloc **word** est une chaine de caractères délimité par un espace (**space**), une tabulation (**tab**) ou la touche retour (**return**) ou bien encore, mise entre guillemets.

Un bloc **word** peut inclure des caractères, des jetons mais pas des items ou des lignes. Il peut-être contenu dans une ligne ou un item, mais pas dans un jeton ou un caractère.

6.1.6 Le Bloc item et la propriété itemDelimiter

Par défaut un item est une chaîne de caractères délimitée par des virgules.

Les items sont délimités par un caractère spécifié dans la propriété **itemDelimiter**. Vous pouvez changer la virgule par défaut pour créer votre propre type de bloc en définissant la propriété **itemDelimiter** à n'importe quel caractère. Un item peut contenir des caractères, des jetons ou des mots, mais pas de lignes. Les items peuvent être contenus dans des lignes, mais pas dans un mot, un jeton ou un caractère.

6.1.7 Le Bloc ligne et la propriété lineDelimiter

Par défaut, une ligne est une chaîne de caractères délimitée par le caractère de retour (**return**) Les lignes sont délimités par le caractère de la propriété **lineDelimiter**. Par défaut, **lineDelimiter** est réglé sur **return**, mais vous pouvez créer votre propre type de bloc en définissant la propriété **lineDelimiter** avec n'importe quel caractère. Une ligne peut contenir des caractères, des jetons, des mots ou des articles. Les lignes ne peuvent pas être contenues dans tout autre type de bloc.

6.1.8 Le Bloc Jeton

Un jeton (**token**) est une chaîne de caractères délimitée par certains signes de ponctuation. Le bloc jeton **(token)** est utile dans l'analyse des déclarations de LiveCode, et est généralement utilisé seulement pour l'analyse des scripts. Pour plus de détails sur la définition du bloc jeton (**token**), voir le Dictionnaire LiveCode. Un jeton (**token**) peut contenir des caractères mais pas n'importe quel autre type de bloc. Les jetons **(token)** peuvent être contenus dans un mot, un élément ou ligne, mais pas dans un caractère.

6.1.9 Spécification d'un Bloc

L'expression de bloc le plus simple définit un seul bloc de n'importe quel type. Les déclarations suivantes sont toutes des expressions de bloc valides :

***get** char 2 of "ABC" -- donne "B"*
get word 4 of "This is a test" **-- *donne* "test"**
get line 7 of myTestData
put "A" into char 2 of myVariable

Vous pouvez également utiliser les nombres ordinaux premier, dernier, milieu, deuxième, troisième, quatrième, cinquième, sixième, septième, huitième, neuvième, dixième (**first**, **last**, **middle**, **second**, **third**, **fourth**, **fifth**, **sixth**, **seventh**, **eighth**, **ninth**, et **tenth**) pour désigner des morceaux simples. L'ordinal spécial **any** définit un bloc aléatoire (random).

```
put "7" into last char of "1085" -- donne "1087"
```

6.1.10 Index Négatifs dans les Expressions de Blocs

Pour compter à rebours à partir de la fin de la valeur au lieu d'avancer depuis le début, spécifier un nombre négatif. Par exemple, le nombre -1, indique le dernier bloc du type spécifié, -2 indique le prochain avant-dernier bloc, et ainsi de suite. Les déclarations suivantes sont toutes des expressions de bloc le valides :

```
get item -1 of "feather, ball, cap" -- donne "cap"
get char -3 of "ABCD" -- donne "B"
```

6.1.11 Complex Chunk Expressions

Les expressions de bloc les plus complexes peuvent être construites en spécifiant un bloc dans une autre bloc. Par exemple, l'expression du bloc word 4 of line 250 indique le quatrième mot de la ligne 250.
En combinant blocs de différents types pour construire une expression de bloc complexe, vous devez spécifier les types de blocs dans l'ordre. Les déclarations suivantes sont toutes des expressions de bloc valides :

```
char 7 of word 3 of myValue
word 9 of item 2 of myValue
last char of word 8 of line 4 of myValue
```

Ceux-ci, cependant, ne sont pas des expressions de bloc valides :

```
word 8 of char 2 of myValue –Les caractères ne peuvent pas contenir de mots
item 9 of first word of myValue –Les mots ne peuvent contenir des items
line 3 of last item of myValue –Les items ne peuvent contenir des lignes
```

6.1.12 Utilisation des Parenthèses dans les Expressions de Bloc

Vous utilisez des parenthèses dans les expressions de bloc pour les mêmes raisons qu'elles sont utilisées dans l'arithmétique :

- pour rendre plus claire une expression complexe,
- pour changer l'ordre dans lequel les parties de l'expression sont évaluées.

Par exemple, considérons la déclaration suivante:

```
put item 2 of word 3 of "a,b,c   i,j,k   x,y,z" -- Mauvais !
```

Le résultat souhaité est "y", le second élément dans le troisième mot. Mais la déclaration ci-dessus provoque une erreur d'exécution, car elle demande un item d'un mot, et les mots ne peuvent pas contenir des items. Vous pouvez obtenir le résultat souhaité en utilisant des parenthèses pour modifier l'ordre d'évaluation :

```
put item 2 of (word 3 of "a,b,c   i,j,k   x,y,z") -- Ok
```

Dans l'exemple ci-dessus, LiveCode obtient le troisième mot d'abord, puis obtient le deuxième item de ce mot. En ajoutant des parenthèses autour (word 3 of "a, b, ci, j, kx, y, z"), vous forcez LiveCode à évaluer la partie de l'expression de ce bloc, d'abord.

La valeur de l'expression entre parenthèses est «x, y, z », et l'item 2 de « x, y, z » est « y ».

Comme pour les expressions arithmétiques, les parties d'une expression du bloc qui sont entre parenthèses sont évaluées en premier. Si les parenthèses sont imbriquées, l'expression du bloc le plus profond entre parenthèses est évaluée en premier. La partie qui est entre parenthèses doit être une expression de bloc valide, ainsi qu'être partie d'une expression de bloc plus large :

put line 2 of word 1 to 15 of myValue **-- Ko**
put line 2 of word (1 to 15 of myValue) **-- Ko**
put line 2 of word 1 to 15 (of myValue) **-- Ko**
put line 2 of (word 1 to 15 of myValue) **-- Ok**

Le premier des exemples ci-dessus ne fonctionne pas pour la même raison que l'exemple précédent : les mots ne peuvent pas contenir de lignes. Les deuxième et troisième exemples ne fonctionnent pas, car ni ***1 to 15 of myValue***, ni ***myValue*** ne sont une expression de bloc valide. Cependant, ***word 1 to 15 of myValue*** est une expression valide de bloc, donc, le dernier exemple fonctionne.

6.1.13 Blocs Inexistants

Si vous demandez un numéro de bloc qui n'existe pas, l'expression du bloc l'évalue comme vide.
Par exemple, l'expression char 7 of "AB" donne vide.
Si vous tentez de modifier un bloc qui n'existe pas, ce qui se passe dépend du type de bloc vous indiquez :

- **Caractère ou un mot qui n'existe pas :** Mettre du texte dans un caractère ou un mot inexistant ajoute le texte à la fin du conteneur, sans insérer d'espaces supplémentaires.
- **Item inexistant :** Mettre du texte dans un objet inexistant, ajoute suffisamment de caractères ***itemDelimiter*** pour faire exister l'élément spécifié.
- **Ligne inexistante :** Mettre du texte dans une ligne inexistante, ajoute suffisamment caractères de retour, pour amener le nombre de ligne spécifié à exister.

6.1.14 Définition d'une plage

Pour spécifier une plus grande partie qu'un seul bloc, vous spécifiez le début et la fin de la plage. Ce sont toutes des des expressions de bloc valides :

get char 1 to 3 of "ABCD" **-- donne "ABC"**
get word 2 to -1 of myValue **-- du second mot au dernier**
put it into line 7 to 21 of myValue **-- remplacement**

Le début et la fin de la plage doivent être indiquées comme le même type de bloc, et la valeur de début de la plage doit être inférieure à celle de la fin. Celles-ci ne sont pas des expressions de bloc valides :

char 3 to 1 of myValue **-- Ko, la fin ne peut pas être supérieure au début**
char -1 to -4 of myValue **-- Ko, le 4ème depuis la fin vient avant dernier**

Important : Lorsque vous utilisez des nombres négatifs dans une gamme, n'oubliez pas que numériquement, -x vient après -x +1. Par exemple, -1 est supérieur à -2, -4 et est supérieur à -7. Le plus grand nombre doit venir en dernier, afin de créer une plage valide.

6.1.15 Compter le nombre de mots, de lignes ou d'items

La fonction **number**, renvoie le nombre de blocs d'un type donné dans une valeur. Par exemple, pour connaître le nombre de lignes dans une variable, utiliser une expression telle que :

```
the number of lines in myVariable
```

Vous pouvez également imbriquer les expressions de bloc pour trouver le nombre de blocs, en un seul bloc, d'un type de bloc plus large :

```
the number of chars of item 10 of myVariable
```

6.2 Comparer et Rechercher

LiveCode fournit un certain nombre de moyens de comparaison et de recherche de texte. Pour la plupart des types de recherche et de comparaison, vous trouverez les expressions de bloc faciles et pratiques. Toutefois, si vous avez des besoins recherche complexes, vous préférerez peut-être utiliser des expressions régulières, couvertes dans la section suivante.

6.2.1 Vérifier si une partie est dans un ensemble

Vous utilisez l'opérateur **is in** pour vérifier si un texte ou de données se situe dans un autre partie du texte ou de données. Vous pouvez utilisez l'opérateur inverse **is not in** pour vérifier si le texte ou de données n'est pas dans une autre partie du texte ou de données.

"A" is in "ABC" **-- évalué à vrai**
"123" is in "13" **-- évalué à faux**

Vous pouvez également utiliser l'opérateur **is in** pour vérifier si un texte ou des données sont dans un bloc spécifié d'un autre conteneur.

"A" is in item 1 of "A,B,C" **-- évalué à vrai**
"123" is in word 2 of "123 456 789" **-- évalué à faux**

6.2.2 Sensibilité à la Casse

Les comparaisons dans LiveCode sont insensibles à la casse par défaut (sauf pour les expressions régulières, qui ont leur propre syntaxe pour spécifier si oui ou non une correspondance doit être sensible à la casse). Pour faire une comparaison sensible à la casse, activez la propriété **caseSensitive**. Pour plus de détails, consultez la propriété **caseSensitive** dans le Dictionnaire de LiveCode.

6.2.3 Vérifier si le texte est Vrai, Faux, un Nombre, un Entier, un Point, un Rectangle, une Date ou une Couleur

Utilisez l'opérateur **is a**, pour vérifier si l'utilisateur a entré les données correctement et pour la validation des paramètres avant de les envoyer à un gestionnaire. L'opérateur **is an** est équivalent à l'opérateur **is a**.

Une valeur est un (***is a***) :

- **boolean** (booléen) si la valeur et vraie ou fausse,
- **integer** (entier) si elle se compose de chiffres (avec un signe moins optionnel),
- **number** (nombre) si elle se compose de chiffres, signe moins optionnel, point décimal optionnel et facultatif "E" ou "e" (notation scientifique),

- **point** si elle est constituée de deux nombres séparés par une virgule,
- **rect** si elle est constituée de quatre nombres séparés par des virgules,
- **date** si elle est dans l'un des formats obtenus par les fonctions de date ou d'heure,
- **color** (couleur) s'il s'agit d'une référence de couleur valide.

Le texte que vous vérifiez peut contenir au début ou à la fin des caractères blancs dans tous les types sauf booléen. Par exemple :

```
" true" is true -- évalué à faux
```

L'opérateur **is a** est l'inverse logique de l'opérateur **is not**.
Quand l'un est vrai, l'autre est faux

```
"1/16/98" is a date -- vrai
1 is a boolean -- faux
45.4 is an integer -- faux
"red" is a color -- vrai
```

Astuce : Pour contraindre un utilisateur à taper des nombres dans un champ, utilisez le script suivant :

```
on keyDown pKey
if pKey is a number then
   pass keyDown
end keyDown
```

Astuce : Le message **keyDown** ne sera transmis que si la touche actionnée par l'utilisateur est un nombre. Si vous piégez un message **keyDown** et qu'il ne passe pas, la saisie ne sera pas entrée dans le champ. Pour plus de détails, voir le message **keyDown** dans le Dictionnaire du LiveCode.

6.2.4 Vérifier si un Mot, un Item ou une Ligne Correspondent Exactement

L'opérateur de **is among** vous indique si tout un bloc existe exactement au sein d'un grand conteneur.
Par exemple, pour savoir si l'ensemble du mot ***free*** est contenu dans une chaîne plus grande, utilisez l'opérateur **is among** :

```
"free" is among the words of "Live free or die" -- vrai
"free" is among the words of "Unfree world" -- faux
```

Le deuxième exemple est évaluée comme faux puisque, bien que la chaîne ***free*** se trouve dans la valeur, c'est une partie d'un plus grand mot, pas un mot entier.

6.2.5 Vérifier si une chaîne commence ou se termine par une autre

Pour vérifier si une chaîne commence ou se termine par une autre chaine, utilisez les opérateurs binaires **Begins With** ou **Ends With**. Par exemple :

```
“foobar” begins with “foo” -- vrai
“foobar” ends with “bar” -- vrai
line 5 of tList begins with "the"
```

6.2.6 Remplacer du texte

Pour remplacer une chaîne par une une autre, utiliser la commande **replace**. (Si vous voulez que la chaîne de recherche contienne une expression régulière, consultez la section sur la commande **replaceText** ci-dessous à la place.)

```
replace "A" with "N" in thisVariable -- changer A en N
```

Pour supprimer du texte en utilisant **replace**, remplacer une chaîne avec la constante **empty**.

```
replace return with empty in field 1
```

Pour plus de détails, voir la commande **replace** dans le dictionnaire de LiveCode.

6.2.7 Récupération de la position d'un bloc correspondant

Les fonctions **wordOffset**, **offset**, **lineOffset**, **itemOffset** peuvent être utilisées pour localiser la position des blocs dans un grand conteneur. Par exemple, cette expression renvoie quel numéro de caractère correspond à la lettre ***C*** :

```
get offset("C","ABC") -- renvoie 3
```

Les fonctions **lineOffset**, **itemOffset** et **wordOffset** peuvent être utilisées de la même mamnière pour localiser des lignes, des items et des mots au sein d'une plus grande valeur.
Pour vérifier si un objet, une ligne ou un mot correspond exactement en utilisant **Offset**, activez la propriété **wholeMatches**.

6.2.8 Les Blocks Résumé

Une expression de bloc décrit l'emplacement d'une partie de texte dans une chaîne plus longue. Les expressions de bloc peuvent décrire des caractères, des items, des jetons, des mots et des lignes de texte.
Pour compter à rebours à partir de la fin d'une chaîne, utilisez des nombres négatifs. Par exemple, le mot -2 indique l'avant-dernier mot. Vous pouvez combiner les expressions de bloc pour spécifier un bloc qui est contenu dans une autre bloc, comme dans le mot 2 de la ligne 3 dans **myVariable**.
Pour la plage d'un bloc, spécifier les points de début et de fin de la plage, comme la ligne 2 à 5 du **myVariable**.
Pour vérifier si un bloc est dans un autre, utilisez l'opérateur **is in**. Pour vérifier si un bloc est un type de données, utilisez l'opérateur **is a**. Pour vérifier si un bloc commence ou se termine par un autre bloc, utiliser les opérateurs **begins with** ou **ends with**. Pour vérifier si un bloc est contenu exactement dans une chaîne utiliser l'opérateur **is among** . Pour obtenir un indice indiquant où un bloc peut être trouvé dans un conteneur, utiliser les fonctions de **wordOffset**, **offset**, **lineOffset**, **itemOffset**. Pour faire correspondre seulement un bloc complet dans une chaîne, régler les **wholeMatches** sur vrai avant d'utiliser l'offset.

6.3 Les Expressions Régulières

Les expressions régulières vous permettent de vérifier si un modèle est contenu dans une chaîne. Utilisez des expressions régulières lorsque l'un des blocs de recherche ou de comparaison ne fait pas ce dont vous avez besoin (voir la section sur la comparaison recherche ci-dessus).

LiveCode supporte la recherche d'un modèle, son remplacement ou le filtrage des lignes dans un conteneur en fonction de ce que, oui ou non, chaque ligne contient ce modèle. Les expressions régulières utilisent la syntaxe PERL compatible ou «PCRE».
Pour plus de détails sur la syntaxe prise en charge, reportez-vous

au cours sur ce site : http://www.regular-expressions.info/examples.html
ou ici : http://perldoc.perl.org/perlre.html
au manuel PCRE à http://www.pcre.org/man.txt

6.3.1 Recherche à l'aide d'une expression régulière

Utilisez la fonction **matchtext** pour vérifier si une chaîne contient un modèle spécifié.

matchtext (string, regularExpression [foundTextVarsList]).

string est une expression, qui renvoie une chaîne.

regularExpression est une expression qui renvoie à une expression régulière.

L'option **foundTextVarsList** consiste en un ou plusieurs noms de variables existants, séparés par des virgules.

```
matchText("Goodbye","bye") -- renvoie vrai
matchText("Goodbye","^Good") -- renvoie aussi vrai
matchText(phoneNumber,"([0-9]+)-([0-9]+-[0-9]+)",areaCode,phone)
```

Pour plus de détails sur cette fonction voir la fonction **matchtext** dans le dictionnaire de LiveCode.

Si vous avez besoin de récupérer les positions des chaînes reconnues dans l'option **foundTextVarsList**, utilisez la fonction **matchChunk** plutôt que la fonction **matchText**.
Ces fonctions sont par ailleurs identiques.

6.3.2 Remplacer en utilisant une expression régulière

Utilisez la fonction de **replaceText** pour rechercher une expression régulière et remplacer les parties qui correspondent. Si vous voulez simplement pour remplacer le texte sans l'aide d'une expression régulière, voir la commande de remplacement à la place.

```
replaceText (stringToChange MatchExpression replacementString)
```

stringToChange est une expression qui renvoie une chaîne.
MatchExpression est une expression régulière.
replacementString est une expression qui renvoie une chaîne.

```
replaceText("malformed","mal","well")--retourne "wellformed"
replaceText(field "Stats",return,comma)-- rend une séparation par virgule
```

Pour plus de détails, voir la fonction de **replaceText** dans le dictionnaire de LiveCode.

Syntaxe des expressions régulières

[chars]	correspond à l'un des caractères à l'intérieur des crochets	A [BCD] E correspond à "ACE", mais pas "AFE" ou "AB"
[^chars]	correspond à n'importe quel caractère qui n'est pas à l'intérieur des crochets	[^ Abc] correspond à "FD" ou "ZD", mais pas "AD" ou "CD"
[char-char]	correspond à la plage du premier au second caractère. La valeur ASCII du premier doit être inférieure à La valeur ASCII de second caractère	A [B-D] correspond à «AB» ou «AC», mais pas «AG» [A-Z0-9] correspond à n'importe quel caractère alphanumérique
.	correspond à n'importe quel caractère unique (sauf un saut de ligne)	A.C correspond à "ABC" ou "AGC", mais pas "AC" ou "ABCD"
^	correspond au caractère qui suit le début de la chaîne (commence par...)	^ A correspond à "ABC" mais pas "CAB"
$	correspond au caractère précédent, à la fin d'une chaîne (finit par...)	B $ correspond à "CAB" mais pas "BBC"
*****	correspond à zéro ou un caractère précédent ou un modèle	ZA * B correspond à «ZB» ou «ZAB» ou «Zaab" mais pas "ZXA" or "AB" [A-D]*G matches "AG" or "G" or "CAG", but not "AB"
+	correspond à une ou plusieurs occurrences du caractère précédent ou d'un modèle	ZA + B correspond à «ZAB» ou «Zaab", mais pas "ZB" [AD] + G correspond à "AG" ou "CAG", mais pas "G" ou "AB"
?	Correspond à zéro ou une occurrence du caractère précédent ou d'un modèle	ZA? B correspond à «ZB» ou «ZAB», mais pas «Zaab" [AD]? G correspond à «AG» ou «CAGZ", mais pas "G" ou "AB"
|	correspond au modèle avant ou après l'|	A | B correspond à "A" ou "B" [ABC] | [XYZ] correspond à «AY» ou «CX», mais pas «AA» ou «ZB».
****	Le caractère suivant doit correspondre littéralement même si cela a une signification spéciale dans une expression régulière	A \. C correspond à "A.C", mais pas "A \. C" "ABC" \ \ correspond à "\"
any other character	correspond à lui-même	ABC matches "ABC"

6.3.3 Filtrage à l'aide d'une Expression Joker

Utiliser la commande **filter** pour enlever des lignes d'un conteneur qui correspond, ou pas, à une expression joker. Les expression joker sont similaires aux expressions régulières.

filter container {with | without} wildcardExpression
filter myVariable with "A?2"
filter me without "*[a-zA-Z]*"

Pour plus de détails, y compris le format des expressions joker, voir la commande de filtre dans le dictionnaire de LiveCode.

6.4 Support des Textes Internationaux

LiveCode fournit un certain nombre de méthodes pour travailler avec les textes internationaux. Ceci inclut la capacité à rendre et à modifier le texte **Unicode** et le convertir entre différents types d'encodage. Nous vous recommandons d'examiner comment vous allez mettre en œuvre support des textes internationaux le plus tôt possible dans la conception de votre application.

6.4.1 Les Encodages Texte

Fondamentalement les ordinateurs utilisent des nombres pour stocker les informations, convertissant ces nombres en texte pour être affichés à l'écran. Un encodage de texte décrit quel nombre se transforme en un caractère donné. Il ya beaucoup de différents systèmes d'encodage pour différentes langues. Voici un tableau contenant des exemples d' encodages courants.

Encodages de Texte Communs

ASCII	Un seul octet - Anglais	**ASCII** est un codage à 7 bits, à l'aide d'un octet par caractère. Il comprend l'alphabet romain, les chiffres arabes, la ponctuation occidentale et les caractères de contrôle. Voir http://en.wikipedia.org/wiki/ASCII pour plus d'informations.
ISO8859	Un seul octet	**ISO8859** est une collection de 10 encodages. Ils sont tous 8 bits, en utilisant un octet par caractère. Chacun partage les 128 premiers caractères **ASCII**. Les 80 caractères suivants changent en fonction de la langue à afficher. Par exemple **ISO8859-1** est utilisé en Europe occidentale, tandis que **ISO8859-5** est utilisé pour le cyrillique. NB : LiveCode ne prend en charge que **ISO8859-1**. Vous devez utiliser Unicode pour représenter d'autres langues, en convertissant si nécessaire (voir ci-dessous).
Windows-1252	Un seul octet - Anglais	Ceci est un sur-ensemble de **ISO8859-1**, qui utilise les 48 caractères restants non utilisés dans le jeu **ISO** pour afficher les caractères sur les systèmes Windows. Par exemple les guillemets courbes sont contenus dans cette plage.
MacRoman	Un seul octet - Anglais	MacRoman est un sur-ensemble de l'**ASCII**. Les 128 premiers caractères sont les mêmes. Les 128 suivants sont entièrement réorganisés et n'ont aucun rapport avec soit **Windows-1252** ou **ISO8859-1**. Cependant, alors que la plupart des symboles sont dans des positions différentes beaucoup sont équivalents de sorte qu'il est possible de convertir entre eux.
UTF-16	Double octet – Toutes langues	**UTF-16** utilise généralement deux octets par point de code (caractère) pour afficher du texte dans toutes les langues du monde (voir l'introduction à **Unicode**, ci-dessous). **UTF-16** prendra plus de mémoire par caractère qu'un codage à octet simple et est donc moins efficace pour l'affichage en anglais.
UTF-8	Multi-octet – Toutes langues	**UTF-8** est un encodage multi-octets. Il a l'avantage que le format **ASCII** y est préservé. Lors de l'affichage

		d'autres langues, **UTF-8** combine ainsi plusieurs octets pour définir chaque point de code (caractère). L'efficacité de l'**UTF-8** dépend de la langue que vous essayez d'afficher. Si vous affichez **Occidental Europe**, il faudra, en moyenne 1,3 octets par caractère, pour la Russie, 2 octets (équivalent à **UTF-16**), mais pour **CJK** 3 ou 4 octets par caractère.

6.4.2 Ecrire des textes

Un script (non informatique) est une façon d'écrire une langue. Il prend l'encodage d'une langue et la combine avec son alphabet pour le rendre à l'écran comme une séquence de glyphes. La même langue peut parfois être écrite avec plusieurs scripts (commun entre les langues de l'Inde). Les scripts peuvent souvent être utilisés pour écrire plus d'une langue (commun entre les langues européennes).

Les scripts peuvent être regroupés en quatre classes approximatives.
La classe de script **small** contient un petit alphabet avec un petit jeu de glyphes pour représenter chaque caractère unique. La classe de script **large** contient un grand alphabet et avec un plus grand ensemble de glyphes. La classe de script **contextual** contient des caractères qui peuvent changer d'apparence en fonction de leur contexte. Et enfin la classe de script **complex** contient des caractères qui sont une fonction complexe du contexte du caractère - il n'y a pas une correspondance 1 à 1 entre le point de code et glyphe.

Roman	Small script	L'encodage Roman a relativement peu de caractères distincts. Chaque caractère a une façon unique d'être écrit. Il est écrit de gauche à droite de haut en bas. Chaque caractère a un glyphe unique. Les caractères ne se joignent pas lors de l'écrit. Par exemple : The quick brown fox.
Chinese	Large script	Le codage chinois a un grand nombre de caractères distincts. Chaque caractère a une façon unique d'être écrit.
Greek	Contextual script	Chaque caractère sauf sigma a un glyphe unique. Sigma change selon que l'on est à la fin d'un mot ou non. Les caractères ne se joignent pas lors de l'écrit. Le texte s'écrit de gauche à droite de haut en bas. Par exemple : Τηε θιψκ βροςν φοχ
Arabic	Contextual script	Le glyphe choisi dépend de sa position dans un mot. Tous les caractères ont des glyphes initiaux, médians et terminaux. Il en résulte un style affichage de style calligraphique. Le texte s'écrit de droite à gauche, de haut en bas de l'affichage. Par exemple : لإات ضعهؤن لاقخصى بخء
Devanagari	Complex script	Dans ce script, il n'y a pas de mappage direct de caractère à glyphe. Les séquences de glyphes se combinent en fonction de leur contexte. Le texte se lit de gauche à droite, de haut en bas.

6.4.3 Introduction l'Unicode

Le but de l'Unicode est de fournir un moyen d'afficher toutes les langues du monde. Auparavant, plusieurs encodages Unicode ont été nécessaires pour afficher du texte à partir de différentes parties du monde (voir ci-dessus). Pour utiliser du texte international dans LiveCode, il est nécessaire d'utiliser Unicode.

6.4.4 Comment fonctionne l'Unicode ?

Unicode est une norme internationale (voir http://unicode.org/). Il fonctionne en attribuant un numéro unique à chaque caractère dans toutes les langues du monde et fonctionne indépendamment de la plate-forme, du programme ou de la langue utilisée. Afin d'avoir suffisamment d'espace pour fournir un numéro unique à chaque caractère, Unicode dispose d'un espace de 21 bits de points de code. La plupart des caractères (mais pas tous) ont une unité de code 1-1 à la cartographie de caractères. Chaque unité de code possède des propriétés implicites pour aider au traitement généralisé. La plupart des langues ont leur propre plage de valeurs spécifique de points de code.
Tous les encodages de texte existants peuvent passer par aller-retour vers Unicode, ce qui signifie qu'ils peuvent être convertis en Unicode et puis de nouveau dans leur format d'origine sans perdre de données. LiveCode a un certain nombre de fonctions que vous pouvez utiliser pour encoder et décoder Unicode. Unicode n'est pas un codage en tant que tel, puisque il peut être représenté dans plusieurs encodages différents (voir le tableau ci-dessus) - **UTF-8**, **UTF-16** ou UTF-32.
La norme Unicode définit les algorithmes de mappage, de tri, de recherche et de césure de mots. LiveCode a actuellement un soutien limité à ces fonctions, mais peut être facilement étendu en utilisant des plugins externes.

6.4.5 Utilisation d'Unicode dans les champs et les objets de LiveCode

Les champs de LiveCode et autres contrôles utilisent l'encodage **UTF-16** pour Unicode. Pour utiliser Unicode dans un champ ou dans les intitulés des commandes, collez le texte Unicode ou réglez le contrôle **textFont** sur "unicode".

Actuellement les champs de LiveCode supportent l'affichage de scripts complexes, mais ne supportent pas encore l'affichage de droite à gauche du texte. Ce support est prévu à l'avenir.
Utilisez le **unicodeText** pour obtenir et définir le contexte d'un champ en **UTF-16** à l'aide d'un script.

```
set the unicodeText of field 1 to URL "binfile:Chinese.txt" -- affiche un fichier en UTF-16 dans un champ
put the unicodeText of field 1 into tUnicodeText -- récupère le contenu d'un champ lui permettant d'être transformé ou manipulé par un script
```

Pour définir le titre d'une pile à une chaîne **UTF-16** utilisez la propriété **unicodeTitle**.

6.4.6 Manipuler Unicode - Utiliser l'UTF-8

Bien que les champs de LiveCode supportent l'affichage de l'**Unicode** en **UTF-16**, les expressions de bloc de LiveCode n'ont pas connaissance actuellement de l'Unicode. Ce support est prévu à l'avenir. À l'heure actuelle, vous devriez écrire vos propres fonctions pour traiter le texte Unicode.

Si vous souhaitez utiliser des fonctions **charToNum** et **numToChar** pour encoder ou décoder des caractères Unicode (**UTF-16**), vous devez définir la propriété locale **useUnicode** comme vraie.

Important : Nous vous recommandons d'utiliser **UTF-8** dans votre application lorsque vous souhaitez stocker ou manipuler des données Unicode. **UTF-8** a l'avantage de préserver la ponctuation ASCII. Cela signifie que vous pouvez toujours utiliser les expressions de bloc **word**, **item** et **line**, pour manipuler **UTF-8**. Vous ne pouvez pas utiliser le bloc *character*, puisqu'il s'agit d'octets individuels.

UTF-8 fonctionnera en aller-retour à travers l'interface d'un plug-in LiveCode.
Mac OS X et Windows ont un ensemble complet de chaîne compatible Unicode, facilitant la manipulation de ces fonctions.
Vous voudrez peut-être envisager de mettre en place un plug-in si vous souhaitez utiliser ces fonctions pour manipuler **UTF-8**.

6.4.7 Conversion entre codages UTF-16 et les autres

Utilisez la fonction de **uniEncode** pour encoder Unicode comme **UTF-16**. Passer à la fonction de **uniEncode** l'encodage que vous convertissez et il va convertir les données, en supposant que les données soient correctement encodées dans l'encodage que vous avez spécifié.

```
put uniEncode(inputText,"Japanese") into tDisplayText -- convertit Shift Japanese Industrial Standards en UTF-16
```

Pour convertir **UTF-16** en retour en **UTF-8** ou à un autre jeu de caractères, utilisez la fonction de **uniDecode**. Lui passer les données au format **UTF-16** avec l'encodage désiré.

```
put uniDecode(field "Japanese","UTF8") into tUTF8 -- convertit un champ en UTF8
```

6.4.8 Conversion entre MacRoman et Windows-1252

Utilisez les fonctions MacToISO et ISOToMac pour convertir entre MacRoman et Windows 1252.

Astuce : LiveCode effectue automatiquement cette translation sur le contenu des champs et des étiquettes d'objet lorsque une pile est chargée sur une plate-forme différente de celle sur laquelle elle a été enregistrée. Toutefois, les données stockées dans les propriétés personnalisées ne sont pas converties car considérées comme données binaires, ainsi que les données dans des fichiers externes. Vous pouvez donc vouloir les traduire manuellement entre les deux jeux de caractères en utilisant ces fonctions :

```
put ISOToMac(tISOText) into tMacText -- convertir depuis Windows-1252 vers MacRoman
put MacToISO(tMacText) into tMacText -- convertir depuis MacRoman vers Windows-1252
```

6.5 Utiliser les tables, Matrices

6.5.1 Quand Utiliser des tables, Matrices

Chaque un élément de tableau est constamment accessible. Cela se compare favorablement avec d'autres fonctions qui recherchent des informations en comptant à travers une variable depuis le début (par exemple les fonctions **offset**). Si vous considérez un problème qui exige de l'ordinateur de rechercher des données, à plusieurs reprises alors, si l'ordinateur doit commencer au début de la variable, la recherche deviendra plus en plus lente au fur et à mesure.
Chaque élément dans un tableau peut contenir des données de n'importe quelle longueur, ce qui rend plus facile de regrouper les enregistrements qui contiennent longueurs assorties ou caractères de séparation. Les tableaux peuvent contenir des éléments imbriqués. Ils sont ainsi idéaux pour représenter les structures de données complexes comme les arbres et les données **XML** qu'il serait difficile de représenter comme une structure plate.
Chaque sous-ensemble d'un tableau peut être consulté et opéré de façon indépendante. Ceci permet de copier une sous table dans autre table, d'obtenir les clés d'une table imbriquée ou de transmettre une table imbriquée en tant que paramètre dans un appel de fonction.
LiveCode inclut diverses fonctions pour convertir des informations vers et à partir de tableaux et pour les opérations sur le contenu des tableaux.
Ces caractéristiques rendent ces tableaux utiles pour un certain nombre de tâches de traitement de données, y compris les tâches qui impliquent le traitement ou la comparaison de grandes quantités de données.
Par exemple, les tableaux sont idéaux si vous voulez compter le nombre d'occurrences d'un mot spécifique dans une partie du texte.
Il serait possible de mettre en œuvre un tel décompte du nombre de mots en parcourant chaque mot et de vérifier sa présence dans une liste de mots, puis en ajoutant une virgule suivie d'un nombre à cette liste de mots. Une telle méthode est lourde à mettre en œuvre et au fur et à mesure que la liste des mots s'allonge la routine va ralentir car LiveCode doit consulter la liste depuis le début de chaque nouveau mot. En revanche, la

mise en œuvre avec un tableau est simple. Parce que chaque élément dans un tableau peut être nommé à l'aide d'une chaîne de texte, nous pouvons créer un élément pour chaque mot et ajouter du contenu à l'élément. Non seulement le code sera beaucoup plus court mais aussi plus rapide.

```
on mouseUp
 --faire défiler chaque mot en ajoutant chaque instance dans un tableau tWordCount[tWord]
  repeat for each word tWord in field "sample text"
    add 1 to tWordCount[tWord]
  end repeat
  -- combiner letableau en texte
  combine tWordCount using return and comma
  answer tWordCount
end mouseUp
```

Résultat de l'exécution de script le nombre de mots

Exemple avec le texte suivant :	**Valeur résultant de tWordCount :**
Single Line – execute single line and short scripts Multiple Lines – execute multiple line scripts Global Properties – view and edit global properties Global Variables – view and edit global variables	Global,4 Line,3 Lines,1 Multiple,2 Properties,2 Single,2 Variables,2 and,3 edit,2 execute,2 scripts,2 short,1 view,2 –,4

6.5.2 Fonctions sur lestableaux de LiveCode

Ce qui suit est une liste de toutes les fonctions de LiveCode utilisables avec les matrices.
Pour une description complète de chacune de ces fonctions voir l'entrée correspondante dans le dictionnaire de LiveCode. Chaque fonction peut être utilisée dans des **table**s au sein d'un tableau. Au lieu de se référer à la variable **array**, reportez-vous à x[x]

add ajoute une valeur à chaque élément dans un tableau où l'élément est un nombre.
combine convertit le texte en un tableau en utilisant des séparateurs que vous définissez.
customProperties renvoie un tableau des propriétés personnalisées d'un objet.
delete variable supprime un élément d'un tableau.
divide divise chaque élément dans un tableau où l'élément est un nombre.

Par exemple: diviser par 3 la variable tArray :

```
divide tArray by 3
```

Résultat de l'exécution de la commande **divide** sur une variable ***array***

Contenu de la variable tArray	**Valeur résultant de tWordCount :**
A = 1 B = 2 C = 3 D = 4 E = 5	0.333333 0.666667 1 1.333333 1.666667

- **element** mot-clé qui est utilisé dans une boucle de répétition pour boucler chaque élément dans un tableau
- **extents** trouve la rangée minimale et maximale et le nombre de colonnes d'un tableau dont les clés sont des nombres entiers
- **intersect** compare les tableaux, en supprimant des éléments d'un tableau si ils ont pas d'élément correspondant dans l'autre
- **keys** retourne une liste de tous les éléments dans un tableau
- **matrixMultiply** effectue une multiplication matricielle sur deux tableaux dont les éléments sont des nombres et dont les clés sont des numéros séquentiels
- **multiply** multiplie une valeur à chaque élément dans un tableau lorsque l'élément est un nombre
- **properties** retourne un tableau des propriétés d'un objet
- **split** convertit un tableau en texte, vous plaçant des délimiteurs entre les éléments séparés
- **sum** renvoie la somme totale de tous les éléments dans un tableau lorsque l'élément est un nombre
- **transpose** échange l'ordre des clés de chaque élément dans un tableau dont les éléments sont des nombres et dont les clés sont des numéros séquentiels
- **union** combine deux tableaux, en éliminant les doublons

6.6 Codage et décodage

LiveCode comprend un certain nombre de fonctions intégrées pour l'encodage et le décodage des données dans une variété de formats populaires.

6.6.1 Texte Stylé

LiveCode supporte l'encodage et le décodage de texte comme **HTML** et **RTF**. Cette fonction est utile lorsque vous voulez régler les styles de texte par programme, l'importation ou l'exportation de texte avec des informations de style.

Important : À l'heure actuelle, le support de la conversion **HTML** ne vaut que pour les styles que l'objet champ de LiveCode est capable de rendre.

Pour convertir le contenu d'un champ avec des balises compatibles **HTML**, utilisez la propriété **HTMLText**. Cette propriété est documentée en détail dans le dictionnaire de LiveCode. Vous pouvez également définir cette propriété pour afficher un texte stylé dans un champ.

Astuce : Vous pouvez obtenir et définir la propriété **HTMLText** d'un bloc dans un champ, vous permettant d'afficher ou de modifier les attributs de texte sur une section du texte. Par exemple, pour définir le style de texte de la ligne 2 d'un champ en gras :

```
on mouseUp
  put the htmltext of line 2 of field "sample text" into tText
  replace "<p>" with "<p><b>" in tText
  replace "</p>" with "</b></p>" in tText
  set the htmltext of line 2 of field "sample text" to tText
end mouseUp
```

Alors, ce n'est pas aussi simple que d'appliquer directement le style au texte en utilisant :

```
set the textStyle of line 2 of field "sample" to "bold"
```

Cette propriété vous permet de rechercher et remplacer des styles de texte ou d'effectuer plusieurs changements complexes dans le texte sur la base de la correspondance de modèles. Effectuer une série de changements sur **HTMLText** dans une variable puis en mettant le texte d'un champ en une fois peut être plus rapide que la mise à jour du style à plusieurs reprises directement sur le champ.

Utilisez le mot-clé **HTML** avec les fonctions de glisser-déposer et les caractéristiques du Presse-papiers pour effectuer la conversion des données vers et à partir de **HTML** lors de l'échange de données avec d'autres applications. Pour plus d'informations, consultez la section **Drag and Drop** dans le chapitre Ecriture d'une interface utilisateur. Utilisez la propriété **RTFText** et le mot-clé **RTF** pour travailler avec le format **RTF**. Utilisez la propriété **unicodeText** et le mot-clé **Unicode** pour travailler avec Unicode. Pour plus d'informations, consultez la section sur le support du texte international, ci-dessus.

6.6.2 Les URL

Pour coder et décoder des **URL**, utilisez les fonctions **URLDecode** et **URLEncode** . La fonction **URLEncode** rendra tout texte plus sûr à utiliser avec une **URL** (par exemple, il remplacera l'espace avec +). Ces fonctions sont particulièrement utiles si vous envoyez des données vers un formulaire Web en utilisant la commande **POST**, en utilisant la commande **launch URL** ou pour envoyer des courriels en utilisant la fonction **revMail**. Pour plus d'informations, voir le Dictionnaire du LiveCode.

Texte	Résultat de l'URL encodé
"Jo Bloggs" <jo@blogs.com>	%22Jo+Bloggs%22+%3Cjo%40blogs.com%3E

6.6.3 Les Données Binaires Base64 (MIME - Pièces Jointes et Transfert Http)

Pour encoder décoder les données en **base64** (par exemple pour créer une pièce jointe), utilisez les fonctions **base64Decode** et **base64Encode**. Ces fonctions sont utiles partout où vous voulez convertir les données binaires en données texte, et inversement. Pour plus d'informations, voir le Dictionnaire de LiveCode.

6.6.4 Données binaires - Types arbitraires

Utilisez les fonctions de **binaryEncode** et **binaryDecode** pour encoder ou décoder les données binaires. Ces fonctions sont documentées en détail dans le dictionnaire de LiveCode.

6.6.5 Conversion d'un Caractère en Nombre

Pour convertir un caractère en sa valeur correspondante ASCII, et inversement, utilisez les fonctions **charToNum** et **numToChar**

```
put charToNum("a") -- retourne 97
```

Pour convertir les caractères Unicode, définissez sur vrai la propriété locale de **useUnicode**. Pour plus d'informations, consultez la section sur le support du texte international, ci-dessus.

6.6.6 Data Compression Compression des Données

Pour compresser et décompresser des données à l'aide GZIP, utilisez les fonctions **compress** et **decompress**.

Astuce : La routine suivante demande à l'utilisateur de sélectionner un fichier, puis crée une version compressée **gzip** avec l'extension **. gz**, dans le même répertoire que l'original.

```
on mouseUp
  answer file "Sélectionnez un fichier : "
  if it is empty then
    exit mouseUp
  end if
  put it & ".gz" into tFileCompressed
  put compress(URL ("binfile:" & it)) into URL ("binfile:" & tFileCompressed)
end mouseUp
```

6.6.7 Chiffrement

Pour crypter ou décrypter des données utilisez les commandes **encrypt** et **decrypt**. Ces commandes sont documentées dans le dictionnaire de LiveCode

6.6.8 Générer une Somme de contrôle (Checksum)

Utilisez le **MD5Digest** pour générer une somme de certaines données. Utilisez cette fonction plus tard pour déterminer si les données ont été modifiées ou pour vérifier que la transmission de l'information était complète.

Astuce : Dans cet exemple, nous enregistrons le **MD5Digest** d'un champ lorsque l'utilisateur ouvre pour l'édition. À l'endroit du script du champ :

```
on openField
  set the cDigest of me to md5Digest(the htmlText of me)
end openField
```

Astuce : Si le champ est modifié (notamment si un style de texte est modifié n'importe où), alors un contrôle ultérieur via **MD5Digest** retournera un résultat différent. Dans l'exemple suivant, nous vérifions cette somme pour déterminer si oui ou non il faut faire apparaître une boîte de dialogue d'alerte à l'utilisateur pour enregistrer les modifications :

```
on closeStackRequest
if the cDigest of field "sample text" is not md5Digest(the htmlText of field "sample text") then
  answer "Save changes before closing?" with "No" or "Yes"
  if it is "Yes" then
    save this stack
  end if
end if
end closeStackRequest
```

6.7 XML

Extensible Markup Language ou **XML**, est un langage d'usage général pour l'échange de données structurées entre les différentes applications et sur Internet. Il se compose de documents de type texte qui sont organisés en structure arborescente. Il peut généralement être lu à la fois par l'homme et la machine.

LiveCode inclut un support complet pour **XML** à travers intégré dans sa bibliothèque **XML**. En outre, des normes existent pour favoriser l'échange de **XML** via une connexion réseau (ou "Web Services") - notamment à travers les protocoles **XML-RPC** et **SOAP**.

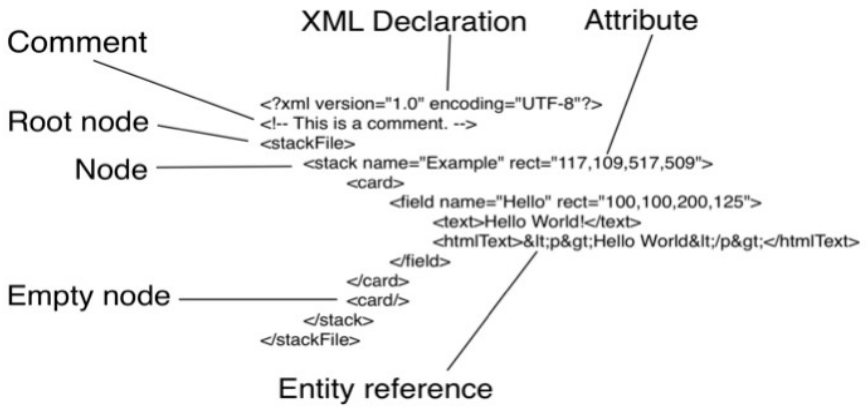
LiveCode comprend une bibliothèque pour l'utilisation de **XML-RPC** et il y a des exemples d'utilisation de LiveCode pour construire des applications **SOAP** disponibles.

6.7.1 L'arborescence XML

XML est simplement un arbre de données. Il doit commencer par un noeud racine, être bien formé et imbriqué. Les étiquettes ne peuvent pas se chevaucher. Pour plus d'informations sur **XML** voir http://en.wikipedia.org/wiki/**XML**.

Dans cet exemple, nous avons représenté un fichier de pile simple, en **XML**. Le fichier de pile comporte une seule pile de deux cartes. Sur la première carte il y a un champ nommé "Hello" avec le contenu "Hello World!". Il y a une deuxième carte, qui est vide.

Arbre XML : Représentation d'une pile



Root node	L'élément racine, élément de document	Le début du document **XML**, qui inclut une déclaration : le fichier est au format **XML**, la version de **XML** l'utilisation et l'encodage du texte.
Comment		Les commentaires peuvent être placés n'importe où dans l'arborescence. Ils commencent par **<!** et se terminent par **>**. Ils ne doivent pas contenir des tirets doubles **--**
Node	Élément, balise	Les éléments qui composent le contenu d'un document **XML**
Attributes		Propriétés attribuables à un noeud donné. Un noeud peut avoir zéro ou plusieurs propriétés.
Empty node	Elément vide	Une manière de spécifier un nœud existant, mais vide.
Entity reference		Procédé de spécification de caractères spéciaux. **XML** inclut le support de &, <,>, 'et". Des entités supplémentaires peuvent être définies à l'aide d'un ***Document Type Definition*** ou **DTD**.

6.7.2 Quand utiliser XML

XML a un certain nombre d'avantages et d'inconvénients. Il est surtout utile lors de l'échange de données entre différents systèmes ou applications. Cependant, comme toute méthode de stockage ou de transfert de données, il n'est pas adapté à tous les types d'application.

Les avantages de **XML** sont les suivants : il est basé sur du texte le rendant plus facilement lisible par les humains aussi bien que par des machines, il est auto descriptif, il est basé sur des normes internationales et largement utilisé avec un grand nombre d'éditeurs disponibles, sa structure hiérarchique le rend approprié pour représenter de nombreux types de documents, et il est indépendant de la plateforme.

Les inconvénients sont les suivants : il est parfois moins efficace que le binaire ou même d'autres formes de représentations textuelles des données ; pour les applications simples, il peut-être plus compliqué qu'il n'est strictement nécessaire, et le modèle hiérarchique peut ne pas être adapté à tous les types de données.

Vous pouvez décider que l'utilisation de **XML** est la meilleure solution pour le stockage de données particulières ou les exigences de transmission. Ou vous pouvez travailler sur un projet avec d'autres personnes où l'utilisation de **XML** ou d'un service Web basé dessus, est une exigence.

Cependant, dans de nombreux cas, un format binaire ou une base de données sera plus appropriée. Vous devez prendre en considération la méthode que vous comptez utiliser le plus tôt possible dans la conception de votre application.

6.7.3 Méthodes de traitement XML dans LiveCode

Le plug-in LiveCode comprend une bibliothèque **XML** complète pour travailler avec des documents **XML**. L'utilisation de la bibliothèque **XML** a l'avantage d'inclure la syntaxe et les fonctions nécessaires pour effectuer les opérations courantes sur **XML**.

Mais l'inconvénient est qu'à l'heure actuelle la bibliothèque est implémentée comme une commande externe (**plug-in** intégré à la distribution dans la zone **plug-in** LiveCode) et donc ne bénéficie pas de la syntaxe du moteur natif de LiveCode.

Si vous avez des besoins simples de traitement **XML**, vous préférerez peut-être utiliser le plug-in de LiveCode pour le support d'expression de bloc (**chunk**), de faire l'analyse syntaxique (**parsing**), de faire correspondre (**match**) ou de construire du **XML**.

Pour plus d'informations, consultez la section sur l'utilisation des expressions **chunk**. Toutefois, si vous travaillez avec du **XML** complexe alors la bibliothèque inclut une suite complète de fonctionnalités.

En plus de la bibliothèque **XML**, le **plug-in** LiveCode a une bibliothèque intégrée basée sur un script pour travailler avec **XML-RPC**.

Astuce : Pour voir la liste des commandes pour travailler avec du **XML-RPC**, filtrer le dictionnaire de LiveCode, avec le terme **XML-RPC**.

Astuce : Comment lire les données à partir d'un fichier du **XML** disponible depuis http://lessons.runrev.com/s/lessons/m/4071/l/7011-How-to-read-in-data-from-an-XML-file

6.7.4 La bibliothèque XML: chargement, affichage et de déchargement XML

Cette section traite de l'utilisation de la bibliothèque **XML** en détail.

Mise en route -- Création d'une arborescence XML dans la mémoire
Pour travailler avec un document **XML**, vous commencez par créer une arborescence **XML** de ce document en mémoire. Il ya deux fonctions **revCreateXMLTreeFromFile** et **revCreateXMLTree**. Utilisez le premier pour charger le document **XML** à partir d'un fichier et pour créer un arbre en mémoire, utilisez la dernière pour créer une arborescence **XML** à partir d'une autre source de données comme une variable, un champ ou un téléchargement.

revCreateXMLTree(XMLText, dontParseBadData, createTree, sendMessages)
revCreateXMLTreeFromFile(filePath, dontParseBadData, createTree, sendMessages)

Avec **revCreateXMLTree** le ***XMLText*** est la chaîne contenant le **XML**. Avec **revCreateXMLTreeFromFile** ce paramètre est remplacé par le **filePath** -- le chemin du fichier du document **XML**. Les deux fonctions renvoient une valeur unique -- l'identification de l'arbre qui a été créé.

Important : Pour ces deux fonctions, vous devez spécifier tous les paramètres. Vous devez stocker l'ID retournée par ces fonctions afin d'accéder à l'arborescence **XML** plus tard dans votre script.

- ***dontParseBadData*** spécifie si oui ou non tenter l'analyse du **XML** mal formé. Si ce paramètre est réglé sur vrai alors les données erronées seront rejetées et généreront une erreur au lieu de construire l'arbre en mémoire.
- ***createTree*** spécifie si vous souhaitez créer un arbre en mémoire ou non. Vous souhaitez généralement que cela soit vrai, sauf si vous avez l'intention seulement de lire dans un fichier **XML** pour déterminer si oui ou non il est bien structuré.
- ***sendMessages*** spécifie si oui ou non les messages doivent être envoyés lors de l'analyse du document **XML**. Les messages peuvent être utiles si vous souhaitez mettre en œuvre des

fonctionnalités telles que la barre de progression, un rendu progressif ou traitement progressif des données d'un fichier **XML** volumineux car il est en cours d'analyse. Si vous définissez cette valeur sur vrai, **revXMLStartTree** sera envoyé lorsque l'analyse commence, **revStartXMLNode** sera envoyé quand un nouveau noeud est rencontré, **revEndXMLNode** sera envoyé quand un nœud a été achevée, **revStartXMLData** sera envoyé au début d'un nouveau bloc de données et enfin **revXMLEndTree** sera envoyé lorsque le traitement est terminé.

Récupération des informations d'un arbre XML

Maintenant que vous avez créé votre arborescence **XML** en mémoire (ci-dessus) et stocké l'**ID** de l'arbre, vous pouvez utiliser les fonctions de cette section pour récupérer des informations de l'intérieur de l'arbre.

Important : Le texte que vous allez chercher en utilisant la bibliothèque **XML** de LiveCode sera encodé comme spécifié dans le noeud racine de l'arbre **XML**.

Note : Tous les exemples de cette section supposent que nous avons chargé l'arborescence **XML** représentée dans le schéma **Arbre XML : Représentation d'une pile**, ci-dessus. Nous supposons que vous avez chargé cet arbre en utilisant la fonction revCreate**XML**Tree décrit ci-dessus, et que cette fonction a retourné une valeur de 1 à l'ID de l'arbre.

Récupération du Nœud Racine
Pour récupérer le nœud racine de votre arbre **XML**, utilisez la fonction **revXMLRootNode**.

```
revXMLRootNode (treeID)
```

La fonction **treeID** contient l'identifiant de l'arbre **XML** auquel vous souhaitez accéder. Par exemple, en utilisant la fonction suivante avec l'arbre de l'échantillon décrit ci-dessus :

```
put revXMLRootNode(1) into tRootNode
```

Récupération du Premier Elément Enfant dans un Noeud
Pour récupérer le premier élément enfant, utiliser **revXMLFirstChild**.

```
revXMLFirstChild (treeID, parentNode)
```

Le **parentNode** contient le chemin du nœud que nous voulons récupérer depuis le premier enfant. Les noeuds sont référencés à l'aide d'un format tel que **/** utilisé pour désigner la racine et délimiter les nœuds. Nous pouvons utiliser cette fonction sur notre échantillon **XML** comme suit :

```
put revXMLFirstChild(1,tRootNode) into tFirstChild          -- transmettre le résultat récupéré dans stackFile dans
                                                                tRootNode, à la fonction revXMLFirstChild.
```

```
put revXMLFirstChild(1,"stackFile") into tFirstChild          -- Équivalent
```

*Il en résulte **tFirstChild** contenant : **/stackFile/stack***

Récupération d'une liste des enfants dans un nœud

Pour récupérer une liste des enfants d'un nœud, utilisez **revXMLChildNames**.

```
revXMLChildNames(treeID, startNode, nameDelim, childName, includeChildCount)
```

- ***nameDelim*** est le séparateur séparant chaque nom qui est retourné. Pour obtenir une liste de noms, indiquez **return**.
- ***childName*** est le nom du type d'enfants à lister.
- ***includeChildCount*** vous permet d'inclure le numéro de chaque enfant entre crochets à côté du nom.

Nous pouvons utiliser cette fonction sur notre échantillon **XML** comme suit :

```
put revXMLChildNames(1,"/stackFile/stack", return, "card", true) into tNamesList
```

*Il en résulte **tNamesList** contenant :*
card[1]
card[2]

Récupérer le contenu des enfants d'un nœud

Pour récupérer une liste des enfants d'un nœud, y compris leur contenu, utilisez **revXMLChildContents**.

```
revXMLChildContents(treeID,startNode,tagDelim,nodeDelim,includeChildCount,depth)
```

Voir ci-dessus pour une explication de ***treeID***, ***startNode*** et ***tagDelim***.

- ***nodeDelim*** indique le séparateur séparant le contenu du nœud de son nom.
  La profondeur indique le nombre de générations d'enfants à inclure. Si vous utilisez -1 comme la profondeur alors tous les enfants sont retournés.

L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLChildContents(1, "/stackFile/stack", space, return, true, -1) into tContents
```

*Il en résulte **tContents** contenant :*
card[1]
field[1]
text[1] Hello World!
htmlText[1] Hello World card[2]

Récupération du nombre d'enfants dans un nœud

Pour récupérer le nombre d'enfants d'un noeud **revXMLNumberOfChildren**.

```
revXMLNumberOfChildren(treeID,startNode,childName,depth)
```

Voir ci-dessus pour une explication de ***treeID***, ***startNode***, ***childName*** et de ***depth***.
En utilisant cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLNumberOfChildren(1, "/stackFile/stack", "card", -1) into tContents
```

*Il en résulte **tContents** contenant :* ***2***

Récupération du parent d'un nœud

Pour récupérer le parent d'un nœud utiliser la fonction ***revXMLParent***.

```
revXMLParent(treeID,childNode)
```

Voir ci-dessus pour une explication de ***treeID***.
En utilisant cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLParent(1,"stackFile/stack") into tParent
```

*Il en résulte **tParent** contenant :*
/stackFile

Récupération d'un des attributs d'un nœud

Pour récupérer un attribut depuis un noeud, utilisez **revXMLAttribute**.

```
revXMLAttribute(treeID,node,attributeName)
```

Voir ci-dessus pour une explication de ***treeID*** et ***Node***.

- ***attributeName*** est le nom de l'attribut dont vous voulez récupérer la valeur. En utilisant cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLAttribute(1,"/stackFile/stack","rect") into tRect
```

*Il en résulte **tRect** contenant :*
117,109,517,509

Récupération de tous les attributs d'un nœud

Pour récupérer tous les attributs depuis un noeud, utilisez **revXMLAttributes**.

```
revXMLAttributes(treeID,node,valueDelim,attributeDelim)
```

Voir ci-dessus pour une explication de ***treeID*** et ***Node***.

- ***valueDelim*** est délimiteur séparant le nom de l'attribut de sa valeur.
- ***attributeDelim*** est délimiteur séparant le nom de l'attribut et la paire de valeurs, l'un de l'autre.

En utilisant cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLAttributes(1, "/stackFile/stack/card/field", tab, return) into tFieldAttributes
```

*Il en résulte **tFieldAttributes** contenant :*
*Nom **Hello rect 100,100,200,125***

Récupérer le contenu d'attributs

Pour récupérer le contenu d'un attribut spécifié à partir d'un noeud et ses enfants, utiliser **revXMLAttributeValues**.

```
revXMLAttributeValues(treeID, startNode, childName, attributeName, delimiter, depth)
```

Voir ci-dessus pour une explication de **treeID**, **startNode** et de **depth**.

- ***childName*** est le nom du type de l'enfant devant être recherché. Laissez ce champ vide pour inclure tous les types d'enfants.
- ***attributeName*** est le nom de l'attribut dont les valeurs sont retournées.
- ***delimiter*** est le délimiteur doit être utilisé pour séparer les valeurs retournées.

En utilisant cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLAttributeValues(1, "/stackFile/", , "rect", return, -1) into tRectsList
```

*Il en résulte **tRectsList** contenant :*
117,109,517,509
100,100,200,125

Récupérer le contenu d'un nœud

Pour récupérer le contenu d'un noeud spécifié, utilisez **revXMLNodeContents**.

```
revXMLNodeContents(treeID, node)
```

Voir ci-dessus pour une explication de ***treeID*** et ***node***.
L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLNodeContents(1, "/stackFile/stack/card/field/htmlText") into tFieldContents
```

*Il en résulte **tFieldContents** contenant :*

** Hello World **

Note : Les références d'entité pour les symboles < et > ont été traduits en texte dans ce résultat.

Récupération des Enfants de mêmes parents

Pour récupérer le contenu de la fratrie d'un nœud, utilisez **revXMLNextSibling** et **revXMLPreviousSibling**.

```
revXMLNextSibling(treeID,siblingNode)
revXMLPreviousSibling(treeID,siblingNode)
```

- ***siblingNode*** est le chemin vers le nœud, pour récupérer les enfants de mêmes parents.

L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLPreviousSibling(1, "/stackFile/stack/card[2]") into tPrev
put revXMLNextSibling(1, "/stackFile/stack/card") into tNext
```

Il en résulte ***tPrev*** contenant:
/stackFile /stack /card[1]

Et ***tNext*** contenant :

/stackFile/stack/card[2]

Recherche d'un nœud

Pour rechercher un nœud en fonction d'un attribut, utilisez **revXMLMatchingNode**.

```
revXMLMatchingNode(treeID, startNode, childName, attributeName, attributeValue, depth, [caseSensitive])
```

Voir ci-dessus pour une explication de ***treeID***, ***startNode*** et ***depth***.

- ***childName*** est le nom des enfants que vous souhaitez inclure dans la recherche. Si vous laissez ce champ vide tous les enfants sont recherchés.
- ***attributeName*** est le nom de l'attribut que vous souhaitez rechercher. ***attributeValue*** est le terme de recherche auquel vous souhaitez correspondre.
- ***caseSensitive*** permet éventuellement de spécifier si la recherche doit être sensible à la casse. La valeur par défaut est faux.

L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLMatchingNode(1,"/", ,"name", "Hello", -1) into tMatch
```

*Il en résulte **tMatch** contenant:*
/stackFile/stack/card[1]/field

Récupération d'un contour de l'arbre (ou une de ses porties)

Pour récupérer le contenu d'un noeud spécifié, utilisez **revXMLTree**.

```
revXMLTree(treeID, startNode, nodeDelim, padding, includeChildCount, depth)
```

Voir ci-dessus pour une explication de **treeID**, **startNode**, **includeChildCount** et **depth**.

- ***nodeDelim*** est le délimiteur qui sépare chaque nœud de l'arbre.
- ***return*** pour récupérer une liste de nœuds.
- ***padding*** est le caractère à utiliser pour indenter chaque niveau dans l'arbre.

L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
put revXMLTree(1,"/",return,space,true,-1) into tTree
```

*Il en résulte **tTree** contenant :*

```
stackFile[1]
 stack[1]
  card[1]
   field[1]
    text[1]
    htmlText[1]
  card[2]
```

Récupération de l'arbre au format XML (ou une des parties)

Pour récupérer l'arbre en **XML**, utilisez **revXMLText**. Utilisez cette fonction pour enregistrer le fichier **XML** dans un fichier après l'avoir modifié.

```
revXMLText(treeID, startNode, [formatTree])
```

Voir ci-dessus pour une explication de ***treeID*** et ***startNode***.

- ***formatTree*** indique s'il faut, oui ou non, formater l'arbre retourné avec des caractères de retour et d'espace pour le rendre plus facile à lire par un humain.

L'utilisation de cette fonction sur notre exemple de fichier **XML** comme suit :

```
ask file "Save XML as:"
put revXMLText(1,"/",true) into URL ("file:" & it)
```

Il en résulte dans le fichier spécifié par l'utilisateur :

```
<stackFile>
<stack name="Example" rect="117,109,517,509">
<card>
<field name="Hello" rect="100,100,200,125">
<text>Hello World!</text>
<htmlText>&lt;p&gt;Hello World&lt;/p&gt;</htmlText>
</field>
</card>
<card/>
</stack>
</stackFile>
```

Validation par rapport à la DTD (Définition du Document Type)

Pour vérifier que la syntaxe d'un fichier **XML** est conforme au **DTD**, utilisez **revXMLValidateDTD**.
Pour plus d'informations sur cette fonction, voir le dictionnaire de LiveCode.

Liste de toutes les arborescences XML en mémoire

Pour générer une liste de tous les arbres **XML** en mémoire, utilisez **revXMLTrees**.
Pour plus d'informations sur cette fonction, voir le dictionnaire de LiveCode.

Retrait d'une arborescence XML la mémoire

Pour supprimer une arborescence **XML** de la mémoire, utilisez **revDeleteXMLTree**.
Pour supprimer tous les arbres **XML** la mémoire, utilisez **revDeleteAllXMLTrees**. Les deux fonctions prennent un seul paramètre - l'identification de l'arbre qui doit être supprimé. Vous devriez supprimer un arbre quand vous avez cessé de l'utiliser.
Pour plus d'informations sur ces fonctions, voir le dictionnaire de LiveCode.

Attention : Une fois que l'arbre **XML** a été supprimé de la mémoire, il n'existe aucun moyen de le récupérer. Utilisez la fonction rev**XML**Text pour récupérer le contenu de l'ensemble de l'arbre et l'enregistrer d'abord.

6.7.5 La bibliothèque XML : édition XML

Cette section explique comment modifier les arbres **XML**. Avant de lire cette section, vous devriez lire la section ci-dessus sur le chargement, l'affichage et le déchargement **XML**.

Ajout d'un nouveau nœud enfant

Pour ajouter un nouveau noeud, utilisez la commande ***revAddXMLNode***.

```
revAddXMLNode treeID, parentNode, nodeName, nodeContents, [location]
```

Voir ci-dessus pour une explication de ***treeID***.

- ***parentNode*** est le nom du noeud auquel vous souhaitez ajouter l'enfant.
- ***nodeName*** est le nom du nouveau noeud à créer.
- ***nodeContents*** est le contenu du nouveau noeud.
- ***location*** - éventuellement spécifier "avant" pour placer le nouvel enfant au départ des nœuds enfants.

Utilisez cette fonction pour ajouter un bouton à notre exemple de fichier **XML** comme suit :

```
revAddXMLNode 1, "/stackFile/stack/card/", "button", ""
```

Il en résulte dans notre arbre un nouveau bouton :

```
<?XML version="1.0" encoding="UTF-8"?> <!-- This is a comment. ->
<stackFile>
<stack name="Example" rect="117,109,517,509"> <card>
<field name="Hello" rect="100,100,200,125"> <text>Hello World!</text>
<htmlText>&lt;p&gt;Hello World&lt;/p&gt;</htmlText>
</field> <button></button>
</card>
<card/> </stack>
</stackFile>
```

Pour créer un nouveau nœud au même niveau que l'autre noeud, utilisez la commande ***revInsertXMLNode*** à la place.

Ajouter un texte XML à une arborescence

Pour ajouter un nouveau noeud, utilisez la commande **revAppendXML**.

```
revAppendXML treeID, parentNode, newXML
```

Voir ci-dessus pour une explication de ***treeID*** et ***parentNode***.

- ***newXML*** est le texte **XML** que vous souhaitez ajouter à l'arbre.

Déplacer, copier ou supprimer un nœud

Pour déplacer un noeud, utilisez la commande **revMoveXMLNode**.

```
revMoveXMLNode treeID, sourceNode, destinationNode [, location] [, relationship]
```

Voir ci-dessus pour une explication de la fonction ***treeID***.

- ***sourceNode*** est le chemin vers le nœud que vous souhaitez déplacer.
- ***destinationNode*** est le chemin vers le nœud vers lequel se déplacer .
- ***location*** indique où le noeud doit être déplacé dans la liste des frères et sœurs - il peut être placé soit avant soit après.
- ***relationship*** vous permet de spécifier si vous voulez placer le nœud à côté de la destination comme un frère ou au-dessous de la destination comme un enfant.

Pour copier un nœud **revCopyXMLNode**.
Pour supprimer un nœud **revDeleteXMLNode**.

Mettre des données dans un nœud

Pour mettre les données dans un noeud, utilisez la commande **revPutIntoXMLNode**.

```
revPutIntoXMLNode treeID,node,newContents
```

Voir ci-dessus pour une explication de la fonction ***treeID*** et ***node***.

- ***newContents*** est le texte que le nouveau noeud contiendra.

Définition d'un attribut

Pour définir un attribut, utilisez la commande **revSetXMLAttribute**.

```
revSetXMLAttribute treeID,node,attributeName,newValue
```

Voir ci-dessus pour une explication de la fonction ***treeID*** et ***node***.

- ***attributeName*** est le nom de l'attribut que vous souhaitez définir l'attribut.
- ***newValue*** est la valeur à définir pour l'attribut.

Utiliser cette fonction pour ajouter une propriété "**showborder**" à notre domaine :

```
revSetXMLAttribute 1, "/stackFile/stack/card/button", "showBorder","true"
```

La balise de champ dans notre arbre ressemble maintenant à ceci :

```
<field name="Hello" rect="100,100,200,125" showBorder="true">
```

Ajout d'une DTD (Definition Document Type)

Pour ajouter une **DTD** à l'arbre, utilisez la commande **revXMLAddDTD**.

```
revXMLAddDTD treeID,DTDText
```

Voir ci-dessus pour une explication de la fonction **treeID**.

- **DTDText** est le texte de la **DTD** à ajouter.

6.8 Tri

Le tri des données est une opération courante et fondamentale. Le tri permet d'afficher des données ou du code d'un certain nombre d'algorithmes de façon conviviale. La fonctionnalité de tri intuitive de LiveCode vous donne la puissance et la flexibilité d'effectuer tout type de tri dont vous pourriez avoir besoin.

6.8.1 Le commandement Sort Container : Aperçu

Pour trier les données, utilisez la commande **sort container**.

```
sort [{lines | items} of] container [direction] [sortType] [by sortKey]
```

- ***container*** est un champ, un bouton, ou une variable, ou la boîte de message.
- ***direction*** est croissant ou décroissant. Si vous ne spécifiez pas direction, le tri est croissant.
- ***sortType*** est l'un des types : ***text***, ***numeric*** ou ***dateTime***. Si vous ne spécifiez pas un ***sortType***, le tri est ***text***.
- ***sortKey*** est une expression qui renvoie une valeur pour chaque ligne ou élément dans le conteneur.

Si la clef de tri contient une sous-chaîne, le mot-clé ***each*** indique que la sous-chaîne est évaluée pour chaque ligne ou un élément. Si vous ne spécifiez pas un ***sortKey***, toute la ligne (ou élément) est utilisé comme ***sortKey***.

L'exemple suivant trie les lignes d'une variable, par ordre alphabétique:

```
sort lines of field "sample text" ascending text
sort lines of tText descending text
```

L'exemple suivant trie une collection d'éléments numériquement:

```
sort items of field "sample csv" ascending numeric
sort items of tItems descending numeric
```

6.8.2 La commande *Sort Container* : Utilisation des clés de tri

La syntaxe ***sortKey*** vous permet de trier chaque ligne ou item sur la base des résultats d'une évaluation effectuée sur chaque ligne ou item.
Pour trier les lignes d'un conteneur par un élément spécifique dans chaque ligne :

```
sort lines of tContainer by the first item of each
sort lines of tContainer by item 3 of each
```

L'expression ***sortKey*** ne sera évaluée qu'une fois pour chaque élément qui doit être trié. Cette syntaxe permet d'effectuer diverses opérations de tri plus complexes.
L'exemple suivant extrait les nombres entiers minimum et maximum présents dans une liste :

```
set the itemDelimiter to "."
sort lines of fld 1 numeric by char 2 to -1 of the first item of each
put char 2 to -1 of the first item of the first line of fld 1 into tMinimumInteger
put char 2 to -1 of the first item of the last line of fld 1 into tMaximumInteger
```

Résultats de commande de tri en utilisant la clé de tri (sortKey)

Liste Originale	Résultat
F54.mov M27.mov M7.mov F3.mov	tMinimumInteger est 3 tMaximumInteger est 54

6.8.3 La commande Sort Container: tri aléatoire

Pour trier aléatoirement, utiliser la fonction ***random*** pour générer un nombre aléatoire en tant que ***sortKey*** pour chaque ligne ou item, au lieu d'évaluer la ligne ou le contenu de l'article.

Par exemple :

```
put the number of lines of tExampleList into tElementCount
sort lines of tExampleList ascending numeric by random(tElementCount)
```

6.8.4 La commande SortContainer : Tri Stable - Tri sur plusieurs clés

Pour trier la liste par de multiples critères, vous pouvez trier plusieurs fois. C'est parce que LiveCode utilise un tri stable, ce qui signifie que si deux articles ont la même clé de tri leur ordre relatif à la sortie ne changera pas. Pour effectuer un tri stable, commencez par les critères les moins signifiants ou importants et travaillez jusqu'aux plus importants ou significatifs.
Par exemple :

```
sort lines of fld 1 ascending numeric by item of each
sort lines of fld 1 ascending text by the first item of each
```

Les résultats de tri d'éléments multiples

Liste Originale	Résultat
Oliver,1.54 Elanor,5.67 Marcus,8.99 Elanor,6.34 Oliver,8.99 Tim,3.44	Elanor,5.67 Elanor,6.34 Marcus,8.99 Oliver,1.54 Oliver,8.99 Tim,3.44

Astuce : Si vous avez un grand ensemble de données et que vous souhaitez améliorer les performances en exécutant seulement un seul tri, vous pouvez construire une clé de tri qui donne l'ordre approprié. Dans cet exemple, une bonne façon de le faire est d'utiliser la fonction de formatage pour construire une chaîne de longueur fixe, un élément par tri :

```
sort lines of fld 1 ascending text by format("%-16s%08.2f", item 1 of each, item 2 of each)
```

Ceci formate chaque ligne individuelle comme les suivantes :

Oliver 00001.54

Elanor 00005.67

L'ordre est déterminé par le second champ, en raison de l'utilisation des caractères de remplissage rendant tous les champs de la même taille.

6.8.5 Tri des Cartes

Pour trier les cartes, utilisez la commande **sort**.

```
sort [marked] cards [of stack] [direction] [sortType] by sortKey
```

stack est une référence à toute pile ouverte. Si vous ne spécifiez pas de pile, les cartes de la pile actuelle sont triées.
direction est croissant ou décroissant. Si vous ne spécifiez pas de direction, le tri est croissant. ***sortType*** est l'un des types : ***text***, ***international***, ***numeric*** ou ***dateTime***. Si vous ne spécifiez pas un ***sortType***, le tri est ***by text***.
sortKey est une expression qui renvoie une valeur pour chaque carte dans la pile. Les références de l'objet au sein de la ***sortKey*** sont traitées comme appartenant à chaque carte en cours d'évaluation, de sorte que

par exemple, une référence à un champ est évaluée selon le contenu de ce champ sur chaque carte. Typiquement, la commande de tri est utilisée avec des champs d'arrière-plan qui ont leur propriété ***sharedText*** sur faux de manière qu'elles contiennent une valeur différente sur chaque carte. Par exemple pour trier les cartes par le contenu du champ ***last name*** de chacun d'elles :

```
sort cards by field "Last Name"
```

Pour trier les les cartes par la valeur numérique d'un code postal:

```
sort cards numeric by field "ZIP code"
```

Astuce : Pour trier les cartes par une expression personnalisée qui effectue un calcul, vous pouvez créer une fonction personnalisée :

```
sort cards by myFunction() -- utilise la fonction ci-dessous

function myFunction
  put the number of buttons of this card into tValue -- exécutez n'importe quel calcul sur tValue ici
  return tValue -- sort utilisera cette valeur
end myFunction
```

Chapitre 7 Programmation d'une interface utilisateur

L'interface utilisateur de votre application est souvent l'une de ses caractéristiques les plus importantes. Dans le chapitre 4, nous avons examiné la façon dont vous construisez une interface utilisateur à l'aide des outils de LiveCode et l'environnement de développement. Dans ce chapitre, nous examinons comment vous pouvez modifier ou même construire une interface utilisateur par programmation. Tout ce que vous pouvez faire en utilisant les outils intégrés vous pouvez aussi le faire par programmation. Vous pouvez même créer et modifier une interface utilisateur au moment de l'exécution dans une application autonome, ou fournir des méthodes interactives pour vos utilisateurs pour modifier certains aspects spécifiques de votre application. Cet ensemble de fonctionnalités vous permettent de produire des applications qui génèrent leur interface à l'aide de fichiers **XML** ou d'une structure de données personnalisée, de construire par programmation des aspects complexes de leur interface, de modifier leur apparence en utilisant les paramètres spécifiés par l'utilisateur, de créer des options permettant des thèmes d'interface ou habillages, de construire votre propre interface de développement pour intégrer vos propres outils dans l'IDE de LiveCode et bien plus encore. Vous pouvez également créer des objets personnalisés et fixer vos propres comportements virtuels et leurs propriétés personnalisées. Nous vous recommandons de passer un peu de temps pour vous habituer avec la construction d'une interface utilisant les outils de l'environnement de développement avant de créer ou de modifier une interface programmation.

7.1 Se Référer aux Objets

En général, vous pouvez vous référer à un objet par son nom, le numéro ou la propriété d'identité.

7.1.1 Se Référer à des Objets par Nom

Vous pouvez vous référer à un objet en utilisant son type d'objet, suivi de son nom. Par exemple, pour faire référence à un bouton nommé **OK** , utilisez l'expression : **bouton "OK"** :

```
set the loc of button "OK" to 32,104
```

Pour modifier le nom d'un objet, entrez un nom dans l'inspecteur de propriétés de l'objet, ou utilisez la commande **set** pour modifier la propriété **name** de de l'objet :

```
set the name of field "Old Name" to "New Name"
select after text of field "New Name"
```

7.1.2 Se Référer à des Objets par Nombre

Un numéro de contrôleur correspond à sa couche sur la carte, de l'arrière vers l'avant. Le numéro de la carte correspond à sa position dans la pile. Un numéro de pile correspond à l'ordre de la création du fichier de pile. Un numéro de pile principale a toujours sa valeur à zéro.

Vous pouvez vous référer à un objet en utilisant son type d'objet suivi de son numéro. Par exemple, pour désigner le troisième champ à partir de celui le plus en arrière, sur une carte, utiliser l'expression, field 3 :

```
set the backgroundColor of field 3 to blue
```

Pour changer le numéro d'une carte ou d'un contrôle, changer le ***Layer box*** dans le volet ***Size & Position*** de l'inspecteur de propriétés de l'objet, ou utilisez le commande **set** pour modifier la propriété ***layer*** de l'objet :

```
set the layer of field "Backmost" to 1
```

Astuce : Les nouveaux objets sont toujours créés au niveau de la couche supérieure. Pour faire référence à un objet que vous venez de créer, utiliser le dernier ordinal :

```
create button
set the name of last button to "My New Button"
```

7.1.3 Se Référer à des Objets par ID (identification)

Chaque objet LiveCode a un numéro d'identification. La propriété d'**ID** ne change jamais (sauf pour les **ID** de pile), et l'**ID** est garanti unique au sein de la pile : il n'y a pas deux objets dans la même pile pouvant avoir la même propriété **ID**.
Vous pouvez vous référer à un objet en utilisant son type d'objet, puis ***keyword ID***, suivie de son numéro d'identification. Par exemple, pour se référer à une carte dont l'***ID property*** est 1154, utiliser l'expression **card ID 1154** :

```
go to card ID 1154
```

Vous ne pouvez pas changer la propriété **ID** d'un objet (à l'exception de la pile).

Important : Autant que possible, vous devez nommer vos objets et se référer à eux par leur nom au lieu d'utiliser un numéro ou **ID**. Le nombre et les propriétés d'identité vont changer si les objets sont copiés et collés. En outre, vos scripts deviendront rapidement difficiles à lire si il y a beaucoup d'**ID** ou références aux objets numériques.

7.1.4 Se Référer à des Objets par Nombre Ordinal

Vous pouvez vous référer à un objet en utilisant son type d'objet, suivi par les nombres ordinaux de **first** (premier) à **tenth** (dixième), ou les ordinaux spéciaux **middle** et **last**. Pour faire référence à un objet aléatoire, utilisez l'ordinal spécial **any**. Par exemple, pour se référer à la dernière carte de la pile actuelle, utilisez **last** :

```
go to last card
```

7.1.5 Le descripteur spécial 'this'

Utilisez le mot-clé pour indiquer la pile actuelle, ou la carte actuelle de la pile:

```
set the backg this roundColor of this stack to white
send "mouseUp" to this card
set the textFont of this card of stack "Menubar" to "Sans"
```

7.1.6 Références des Contrôles

Un contrôle est un objet qui peut apparaître sur une carte. Les champs, boutons, barres de défilement, images graphiques, lecteurs, objets **EPS**, et les groupes sont tous des contrôles. Piles, cartes, clips audio et clips vidéo ne sont pas des contrôles.
Vous pouvez vous référer à un objet de l'un de ces types d'objets, en utilisant le mot **control**, suivi d'un **ID**, d'un nom ou d'un numéro :

```
hide control ID 2566
send mouseDown to control "My Button"
set the hilite of control 20 to false
```

Si vous utilisez un nom, comme dans le contrôle de l'expression "**Thing**", la référence est au premier contrôle (ayant la couche la plus basse) qui porte ce nom.

Lorsque vous faites référence à un contrôle par numéro en utilisant son type d'objet, la référence est le Nième contrôle de ce type. Par exemple, l'expression **field 1**, se réfère au champ le plus bas sur la carte. Cela peut ne pas être le contrôle le plus bas, parce qu'il peut y avoir des contrôles d'autres types sous **field 1**. Toutefois, lorsque vous faites référence à un contrôle par numéro en utilisant le mot **control**, la référence est le Nième contrôle de tout type. L'expression **control 1** se réfère au plus faible contrôle sur la carte, qui peut être de n'importe quel type.

Astuce : Pour faire référence à l'objet sous le pointeur de la souris, utilisez la fonction **mouseControl**.

7.1.7 Références des Objets Imbriqués

Pour faire référence à un objet qui appartient à un autre objet, imbriquez les références dans le même ordre que la hiérarchie des objets. Par exemple, si il y a un bouton appelé "My Button" sur une carte appelée "Ma Card", vous pouvez référer à ce bouton comme ceci :

```
show button "My Button" of card "My Card"
```

Vous pouvez mélanger les noms, numéros, références ordinales et des **ID** dans une référence à un objet imbriqué, et vous pouvez imbriquer des références quelque soit la profondeur nécessaire pour préciser l'objet. La seule exigence est que l'ordre des références soit le même que l'ordre de la hiérarchie des objets, allant d'un objet vers l'objet qui en est propriétaire. Voici quelques exemples :

```
field ID 34 of card "Holder"
player 2 of group "Main" of card ID 20 of stack "Demo" first card of this stack
stack "Dialog" of stack "Main" -- "Dialog" est une sous-pile
```

Si vous ne spécifiez pas de carte en référence à un objet qui est contenu dans une carte, LiveCode assume que l'objet est sur la carte actuelle. Si vous ne spécifiez pas de pile, LiveCode assume que l'objet est dans la pile actuelle. Vous pouvez référencer un contrôle dans une autre pile par l'une des méthodes suivantes :

Utiliser une référence imbriquée qui contient le nom de la pile :

```
field 1 of stack "My Stack"
graphic "Outline" of card "Tools" of stack "Some Stack"
```

Définissez la propriété **defaultStack** pour la pile à laquelle vous voulez faire référence en premier. **defaultStack** spécifie la pile en cours, de sorte que vous pouvez vous référer à n'importe quel objet dans **defaultStack** sans y inclure le nom de la pile.
Cet exemple définit une case à cocher dans la pile en cours pour avoir même valeur que la case à cocher dans une autre pile appelée **Other Stack** :

```
put the defaultStack into savedDefault -- ainsi vous pouvez retrouver le réglage initial
set the defaultStack to "Other Stack"
put the hilite of button "Me" into meSetting -- ce bouton est dans "Other Stack"
set the defaultStack to savedDefault
set the hilite of button "Me Too" to meSetting -- ce bouton est dans la pile d'origine
```

Si un objet est dans un groupe, vous pouvez inclure ou omettre une référence au groupe dans une référence imbriquée à l'objet. Par exemple, supposons que la carte actuelle contient un bouton appelé «Guido», qui fait partie d'un groupe appelé «Stereotypes». Vous pouvez vous référer au bouton avec l'une des expressions suivantes :

```
button "Guido"
button "Guido" of card 5
button "Guido" of group "Stereotypes"
button "Guido" of group "Stereotypes" of card 5
```

S'il n'y a pas d'autre bouton nommé "Guido" sur la carte, ces exemples sont équivalents. S'il y a un autre bouton du même nom dans un autre groupe (ou sur la carte, mais pas dans n'importe quel groupe), vous devez spécifier le groupe (comme dans le deuxième et troisième exemple) ou nommer la le bouton par sa propriété **ID**, pour être sûr que vous faites référence au bouton approprié.

7.2 Propriétés

Une propriété est un attribut d'un objet LiveCode. Chaque type d'objet a de nombreuses propriétés intégrées, qui affectent l'apparence ou le comportement de l'objet. Vous pouvez également définir des propriétés personnalisées pour n'importe quel objet, et les utiliser pour stocker tout type de données.
Cette rubrique explique comment utiliser les propriétés, comment les propriétés sont héritées entre les objets, et comment créer et basculer entre des ensembles de paramètres de propriété.
Pour bien comprendre cette rubrique, vous devez savoir comment créer des objets, comment utiliser l'inspecteur de propriétés d'un objet, et comment écrire des scripts courts.

7.2.1 Utilisation des propriétés de l'objet

Une propriété est un attribut d'objet, et chaque type d'objet a son propre ensemble de propriétés intégrées, approprié. Un objet peut être complètement décrit par ses propriétés intégrées, et si, pour deux objets, vous pouviez rendre toutes leurs propriétés identiques, ils seraient semblables. Il est ainsi possible de décrire un objet tout à fait comme un ensemble de propriétés ou d'exporter et d'importer des propriétés en utilisant un fichier texte ou **XML**. Plus de détails sur quelques-unes des méthodes que vous pouvez utiliser pour faire cela sont traités plus loin dans ce chapitre.

Note : Puisque deux objets ne peuvent avoir la même **ID**, il n'est pas possible, dans la pratique, pour deux objets différents de devenir le même objet, parce que leur **ID** sera toujours différente.

Les propriétés incorporées déterminent l'apparence et le comportement des piles et de leur contenu - polices, couleurs, types de fenêtres, la taille et le placement, et bien plus encore - ainsi que beaucoup du comportement de l'application LiveCode. En changeant les propriétés, vous pouvez modifier presque tous les aspects de votre application. Lorsque vous combinez la capacité de modifier les propriétés avec la possibilité de créer et supprimer des objets par programmation, vous pouvez modifier tous les aspects de votre application.

7.2.2 Référence à de Propriétés

Les références de propriété, se composent du mot **the**, le nom de la propriété, le mot **of**, et une référence à l'objet :

```
the armedIcon of button "My Button"
the borderWidth of field ID 2394
the name of card 1 of stack "My Stack"
```

Les propriétés sont sources de valeur, de sorte que vous pouvez obtenir la valeur d'une propriété en l'utilisant dans une expression :

```
put the height of field "Text" into myVar
put the width of image "My Image" + 17 after field "Values"
if item 1 of the location of me > zero then beep
```

Par exemple, pour utiliser la propriété de la largeur d'un bouton dans le cadre d'une expression arithmétique, utilisez une instruction comme suit :

```
add the width of button "Cancel" to totalWidths
```

La valeur de la propriété - dans ce cas, la largeur du bouton en pixels - se substitue à la référence de la propriété lorsque l'instruction est exécutée.

7.2.3 Modification des propriétés

Pour changer la valeur d'une propriété, vous utilisez la commande set :

```
set the borderColor of group "My Group" to "red"
set the top of image ID 3461 to zero
```

Vous pouvez également afficher et modifier un grand nombre de propriétés d'un objet en sélectionnant l'objet et en choisissant **Object Inspector**.
Voir le chapitre Création d'une interface utilisateur pour plus de détails.

Astuce : Pour voir une liste de tous les termes du langage (y compris les propriétés) applicables à un type d'objet particulier, ouvrir le dictionnaire de LiveCode, faites un clic droit sur la barre d'en-tête pour sélectionner le type d'objet que vous voulez, puis triez la liste en cliquant sur l'entête d'objet .

La plupart des propriétés intégrées affectent l'apparence ou le comportement de l'objet. Par exemple, la hauteur, la largeur et l'emplacement d'un bouton sont les propriétés du bouton. La modification de ces propriétés dans un gestionnaire provoque le changement d'apparence du bouton. Inversement, en faisant glisser ou en redimensionnant le bouton cela modifie les propriétés connexes.

Propriétés en lecture seule

Certaines propriétés peuvent être lues, mais non réglées. Elles sont appelées propriétés en lecture seule. Tenter de définir une propriété en lecture seule provoque une erreur d'exécution. Pour savoir si une propriété est en lecture seule, consultez son entrée dans le dictionnaire de LiveCode.

Modifier une partie d'une propriété

Les propriétés ne sont pas des conteneurs, de sorte que vous ne pouvez pas utiliser une expression de bloc pour modifier une partie de la propriété. Cependant, vous pouvez utiliser une expression de bloc pour examiner une partie d'une propriété. Par exemple, vous ne pouvez pas définir la ligne 1 d'une propriété à une nouvelle valeur : vous devez définir l'ensemble de la propriété.
Pour plus de détails, voir la section Expressions de Blocs dans le chapitre sur le traitement du texte et de données.

Pour modifier une partie d'une propriété, d'abord mettre les valeurs de la propriété dans une variable, changer la partie souhaitée dans la variable, puis redéfinissez la propriété en retour avec les nouveaux contenus de la variable :

```
put the rect of me into tempRect
put "10" into item 2 of tempRect
set the rect of me to tempRect
```

Les propriétés personnalisées et les propriétés virtuelles

Une propriété personnalisée est une propriété que vous définissez. Vous pouvez créer autant de propriétés personnalisées pour un objet que vous voulez, et y mettre tout type de données en elles, y compris des données binaires ou des tableaux. Vous pouvez même stocker un fichier dans une propriété personnalisée.
Les propriétés virtuelles sont des propriétés personnalisées qui déclenchent une action de script personnalisée lorsque vous les modifiez, vous permettant de mettre en œuvre des comportements d'objets "virtuels".
Les propriétés personnalisées et les propriétés virtuelles sont couverts dans leurs sections respectives plus loin dans ce chapitre.

7.2.4 L'héritage de propriété

La plupart des propriétés sont spécifiques à l'objet dont elles font partie, et affectent uniquement cet objet. Cependant, certaines propriétés d'un objet, comme sa couleur et la police du texte, prennent les paramètres de l'objet parent dans la hiérarchie de l'objet. Par exemple, si la propriété de couleur de fond d'un champ n'est pas spécifié (ce qui est, si sa propriété **backgroundColor** est vide), le champ prend la couleur de fond de la carte à laquelle il appartient.
Si aucune couleur d'arrière-plan n'est spécifiée pour la carte, la couleur de fond de la pile est utilisée, et ainsi de suite. Cela signifie que vous pouvez définir une couleur de fond pour une pile, et chaque objet, utilisera automatiquement cette couleur de fond, sans que vous ayez à la définir pour chaque objet.
Ce processus va, d'abord vérifier l'objet, puis le propriétaire de l'objet, l'objet qui possède cet objet, et ainsi de suite, est appelé héritage de propriétés. Chaque objet hérite de la couleur de fond de l'objet au-dessus de lui dans la hiérarchie. Des règles d'héritage similaires s'appliquent à **foregroundColor**, **topColor**, **bottomColor**, **borderColor**, **shadowColor** et **focusColor**, à leurs propriétés de configuration correspondantes, à **textFont**, **textSize** et les propriétés **textStyle**.

7.2.5 Surcharger l'héritage

L'héritage est utilisé pour déterminer l'apparence d'un objet que si l'objet lui-même n'a pas de paramètres de propriété. Si une propriété héritée d'un objet n'est pas vide, ce paramètre remplace tout paramètre dont l'objet pourrait hériter d'un objet parent à lui dans la hiérarchie.
Par exemple, si la propriété **backgroundColor** d'un bouton est réglée sur une référence de couleur au lieu d'être vide, le bouton utilise, cette couleur de fond, indépendamment de l'héritage du bouton. Si l'objet a une couleur qui lui est propre, cette couleur est toujours utilisée.

Le mot-clé *effective*

Si une propriété héritée d'un objet est vide, vous ne pouvez pas vérifier simplement la propriété pour savoir si ce paramètres de couleur ou de police, l'objet s'affichera. Dans ce cas, utilisez le mot-clé effective pour obtenir le réglage hérité de la propriété. Le mot-clé effective recherche les propriétaires de l'objet, si nécessaire, pour savoir si ce paramètre est réellement utilisée.

Par exemple, supposons que vous avez un champ dont la propriété **textFont** est vide. Le **textFont** de la carte supportant le champ est réglé sur "Helvetica", de sorte que le champ hérite de ce paramètre et affiche son texte dans la police Helvetica.
Pour savoir ce que la police du champ utilise, utilisez l'expression **the effective textFont** :

get the textFont of field "My Field" **-- vide**
get the effective textFont of field "My Field" **-- Helvetica**

Vous pouvez utiliser le mot-clé **effective** avec n'importe quelle propriété héritée.

7.3 Propriétés Globales

LiveCode possède également des propriétés globales, qui affectent comportement global de l'application. Les propriétés globales sont accessibles et modifiables de la même manière que les propriétés de l'objet. Ils n'appartiennent pas à un objet particulier, mais sinon, ils se comportent comme des propriétés d'un objet.

Astuce : Pour voir une liste de toutes les propriétés globales, ouvrez la boîte de message, et cliquez sur l'icône **Global Properties** - la troisième icône de la gauche en haut de la fenêtre. Pour voir une liste de toutes les propriétés du langage LiveCode, y compris les propriétés globales et objets, utilisez la propriété globale **propertyNames**.

Quelques propriétés sont à la fois des propriétés globales et d'objet.
Par exemple, **paintCompression** est une propriété globale, mais également une propriété d'image.
Pour ces propriétés, le paramètre global est séparé du paramètre pour objet individuel.
D'autres propriétés globales sont affectées par les paramètres du système. Par exemple, la valeur par défaut de la propriété **playLoudness** est défini par le réglage du volume sonore du système d'exploitation.

7.3.1 Références aux propriétés globales

Vous faites référence à des propriétés globales à l'aide **the** et nom de la propriété :

the defaultFolder
the emacsKeyBindings
the fileType

Puisque les propriétés globales s'appliquent à l'ensemble de l'application, vous n'avez pas à inclure une référence objet pour se référer à eux.

Les propriétés globales sont sources de valeur, de sorte que vous pouvez obtenir la valeur d'une propriété globale à l'aide d'une expression :

```
get the stacksInUse
put the recentNames into field "Recent Cards"
if the ftpProxy is empty then exit setMyProxy
```

7.3.2 Changer les propriétés globales

Pour modifier une propriété globale, vous utilisez la commande **set**, de la même façon que pour les propriétés d'un objet :

```
set the itemDelimiter to "/"
set the grid to false
set the idleTicks to 10
```

Certaines propriétés globales peuvent être modifiées par d'autres commandes. Par exemple, la propriété **lockScreen** peut être réglée soit directement, soit modifiée à l'aide des commandes **lock screen** et **unlock screen**. Les deux instructions suivantes sont équivalentes :

```
set the lockScreen to false -- fait la même chose que... unlock screen
```

7.3.3 Sauvegarde et restauration des propriétés globales

Les propriétés de l'objet font partie de l'objet, de sorte qu'elles sont enregistrées lorsque la pile contenant leur objet est enregistrée. Les propriétés globales, cependant, ne sont pas associées à un objet, de sorte qu'elles ne sont pas enregistrées avec une pile. Si vous modifiez la valeur d'une propriété globale, le changement est perdu lorsque vous quittez l'application.
Si vous voulez utiliser le même réglage de propriété globale au cours d'une autre session de votre application, vous devez enregistrer ce paramètre - dans un fichier de préférences, dans une propriété personnalisée, ou ailleurs dans un fichier enregistré - et le restaurer au démarrage de votre application .

7.4 Propriétés Liées au Texte

Normalement, les propriétés sont appliquées uniquement aux objets ou, dans le cas de propriétés globales, à l'ensemble de l'application. Cependant, quelques propriétés s'appliquent également aux blocs dans un champ ou à des caractères uniques dans un champ.

7.4.1 Propriétés de style de texte

Certaines propriétés liées au texte peuvent être appliquées soit à un champ entier soit à un bloc du champ:

```
set the textFont of word 3 of field "My Field" to "Courier"
set the foregroundColor of line 1 of field 2 to "green"
if the textStyle of the clickChunk is "bold" then beep
```

Les propriétés de champ suivantes peuvent être appliquées soit à tout un champ soit à une partie de du champ :

- **textFont**, **textStyle**, et **textSize**
- **textShift**
- **backgroundColor** et **foregroundColor**
- **backgroundPattern** et **foregroundPattern** (systèmes **Unix**)

Chaque bloc du champ hérite de ces propriétés du champ, de la même façon que les champs héritent de leurs propriétaires. Par exemple, si la propriété **textFont** d'un mot est vide, le mot est affiché dans la police du champ. Mais si vous définissez le mot avec **textFont** réglé sur une autre police de caractères, ce mot - et seulement ce mot - est affiché dans sa propre police.

Pour trouver le style du texte d'un bloc dans un champ, si ce bloc utilise ses propres styles ou les hérite du champ, utilisez le mot clé **effective**:

```
get the effective textFont of word 3 of field ID 2355
answer the effective backgroundColor of char 2 to 7 of field "My Field"
```

Astuce : Si une expression de bloc comprend plus d'un style, la propriété correspondante pour ce bloc indique «mixte». Par exemple, si la première ligne du champ a une textSize de "12", et la deuxième ligne a une textSize de "24", une expression comme la suivante indique "mixed" :

```
the textSize of line 1 to 2 of field "My Field"
```

7.4.2 Propriétés du texte formaté

Les propriétés *formattedRect* et connexes

La propriété **formattedRect** (avec **formattedWidth**, **formattedHeight**, **formattedLeft** et **formattedTop**) indiquent la position d'un bloc de texte dans un champ. Ces propriétés sont en lecture seule.

formattedRect, **formattedLeft** et les propriétés **formattedTop** peuvent être utilisées pour un bloc du champ, mais pas pour tout le champ. **formattedWidth** et **formattedHeight** s'appliquent à la fois aux champs et aux blocs du texte dans un champ.

Les propriétés imageSource, linkText et visited

La propriété **imageSource** d'un caractère spécifie une image pour se substituer à ce caractère lorsque le champ est affiché. Vous utilisez **imageSource** pour afficher des images à l'intérieur de champs :

```
set the imageSource of char 17 of field 1 to 49232
set the imageSource of char thisChar of field "My Field" to "http://www.example.com/banner.jpg"
```

La propriété **linkText** d'un bloc vous permet d'associer du texte masqué avec une partie du texte d'un champ. Vous pouvez utiliser **linkText** dans un gestionnaire **linkClicked** pour indiquer la destination d'un lien hypertexte, ou pour toute autre raison.

La propriété **visited** spécifie si vous avez cliqué sur un groupe de texte lors de la session en cours. Vous pouvez obtenir la propriété **visited** pour n'importe quel bloc dans un champ, mais il n'a de sens que si le textstyle du bloc comprend "***link***".

Les propriétés **imageSource**, **linkText** et **visited** sont les seules propriétés qui peuvent être définies pour un bloc dans un champ, mais pas pour l'ensemble du champ ou pour tout autre objet. Parce qu'ils sont appliqués au texte dans les champs, ils sont répertoriés comme propriétés du champ dans le dictionnaire de LiveCode.

Les propriétés **htmlText**, **RTFText** et **UnicodeText** d'un bloc sont égales au texte de ce bloc, avec les informations de mise en forme qui conviennent à la propriété.

Par exemple, si un champ contient le texte "Ceci **est** un test.", Et le mot "**est**" est en gras, ***the html Text of word 2*** indique **<b>**est**</ b>**.

Pour plus d'informations sur ces propriétés, voir le chapitre sur le traitement du texte et des données, ainsi que les entrées individuelles pour ces propriétés dans le dictionnaire de LiveCode.

7.5 Créer et Supprimer des Objets

LiveCode vous permet de créer et supprimer des objets par programmation. Vous pouvez éventuellement spécifier toutes les propriétés d'un nouvel objet avant de le créer.

7.5.1 La Commande Créer des Objets

Vous utilisez la commande **create** pour créer un nouvel objet.
create [invisible] type [name] [in group]

type est tout contrôle qui peut être mis sur une carte : champ, bouton, image, barre de défilement, graphique, lecteur, ou **EPS**.

Le nom est le nom de l'objet nouvellement créé. Si vous ne spécifiez pas de nom, l'objet est créé avec un nom par défaut.

Le groupe est tout groupe présent sur la carte actuelle. Si vous spécifiez un groupe, le nouvel objet est un membre du groupe, et existe sur chaque carte disposant ce groupe. Si vous ne spécifiez pas un groupe, l'objet est créé sur la carte actuelle et que sur celle-ci.

```
create button "Click Me"
create invisible field in first group
```

Pour plus de détails, voir la commande **create** dans le dictionnaire de LiveCode. Pour plus de détails sur la façon de définir les propriétés d'un objet avant de le créer, voir la section sur la création d'objets hors de l'écran en utilisant le modèle des objets, ci-dessous.

7.5.2 Commande pour Supprimer un Objet

Vous pouvez utiliser la commande **delete** pour supprimer des objets de la pile.
delete {objet}

L'objet est tout objet disponible.

```
delete this card
delete button "New Button"
```

Pour plus de détails, voir la commande **delete** dans le dictionnaire de LiveCode.

7.5.3 Création d'objets hors écran en utilisant des modèles Objets

LiveCode utilise des objets modèles (***template objects***), pour vous permettre de spécifier les propriétés d'un objet avant qu'il ne soit créé. Les objets modèles sont des modèles hors de l'écran - il ya un pour chaque type d'objet possible, par exemple, bouton, champ, graphique, etc.

Si vous devez créer un nouvel objet, puis définir certaines propriétés de l'objet, il est plus efficace de faire les modifications de l'objet modèle, puis créer l'objet. Parce que l'objet peut être créé avec toutes ses propriétés correctement réglées, il n'est pas nécessaire de verrouiller l'écran et de mettre à jour ou de repositionner l'objet après sa création.

Par exemple, l'environnement de développement de LiveCode utilise des objets modèles internes pour créer de nouveaux objets de la palette d'outils principale.

Vous définissez les propriétés d'objets modèles de la même façon que vous définissez les propriétés d'objets normaux.

```
set the {property} of the template{Objecttype} to {value}
```

Par exemple, pour créer un bouton avec le nom "Hello World", positionné à 100,100:

```
set the name of the templateButton to "Hello World"
set the location of the templateButton to 100,100
create button
```

Lorsque vous avez utilisé le **templateObject** pour créer un nouvel objet, vous devez le réinitialiser avant de l'utiliser à nouveau.
Réinitialisation du **templateObject** rétablit toutes ses propriétés, aux valeurs par défaut.

```
reset the template[Objecttype]
```

Par exemple, pour réinitialiser le **templateButton** :

```
reset the templateButton
```

Pour plus de détails sur les objets modèles, recherchez dans le dictionnaire de LiveCode pour "**template**".

7.6 Récupérer les Propriétés dans une Variable Array avec Properties

En plus de récupérer les propriétés d'un objet individuel, vous pouvez récupérer ou définir un ensemble complet dans une table (**array**) en utilisant la propriété **properties**. Vous pouvez l'utiliser pour modifier, copier, exporter ou importer des propriétés.

```
set the properties of object to propertiesArray
put the properties of object into propertiesArray
```

properties pour un objet est une table (**array**) contenant les propriétés intégrées significatives de cet objet.

```
put the properties of button 1 into myArray
set the properties of last player to the properties of player "Example"
```

Astuce : Cet exemple de gestionnaire vous montre comment transcrire la propriété **properties** d'un objet vers un fichier texte.

```
on mouseUp
  put the properties of button 1 into tPropertiesArray
  combine tPropertiesArray using return and "|"
  ask file "Sauvegardez les propriétés comme :"
  if it is not empty then
    put tPropertiesArray into URL ("file:" & it)
  end if
end mouseUp
```

Dans cet exemple, le nom de chaque propriété sera écrit suivi du caractère "|" et la valeur de la propriété, puis un caractère de retour.

Pour plus de détails, consultez la propriété **properties** dans le dictionnaire de LiveCode.

7.7 Profils de propriété

Un profil de propriété est une collection de paramètres de propriétés d'objet, stockée comme un ensemble. Un profil pour un objet peut contenir des paramètres pour presque toutes les propriétés de l'objet. Vous pouvez inclure des valeurs pour la plupart des propriétés intégrées dans un profil, et créer autant de profils différents de propriété que nécessaire pour n'importe quel objet. Une fois que vous avez créé un profil, vous pouvez basculer l'objet vers ce profil pour changer toutes les valeurs de propriétés définies dans le profil. Par exemple, supposons que vous créiez un profil de propriété d'un champ contenant les paramètres pour les propriétés de couleur du champ. Lorsque vous passez à ce profil, les couleurs du champ changent, alors que toutes les autres propriétés (non incluses dans le profil) restent les mêmes.
Utilisez les profils de propriété lorsque vous souhaitez :

- Créer des "skins" pour votre application
- Afficher votre application dans différentes langues
- Présenter différents niveaux - "novices", "expert" et ainsi de suite.
- Utiliser différentes normes d'interface utilisateur pour les les différentes plateformes

Pour plus de détails sur la façon de créer des profils de propriété à l'aide de l'IDE, voir la section sur les profils de propriétés dans le chapitre Construction d'une interface utilisateur.

7.7.1 Noms de Profils

Les noms de profil suivent les mêmes règles que les noms de variables. Un nom de profil doit être un seul mot, composé de lettres, chiffres et caractères de soulignement, et doit commencer par une lettre ou un trait de soulignement.

7.7.2 Le Profil Maître

Chaque objet a un profil maître qui contient les paramètres par défaut pour toutes ses propriétés. Si vous ne réglez pas le profil d'un objet, le profil maître est utilisé. Lorsque vous créez un nouveau profil, vous pouvez modifier les paramètres de diverses propriétés pour les rendre différents des paramètres du profil maître.

Astuce : Si vous souhaitez utiliser une seule commande pour passer plusieurs objets sur un profil particulier, donnez au profil le même nom pour chacun des objets auquel il s'applique.

Si vous ne spécifiez pas un paramètre de propriété dans un profil, le réglage du profil maître est utilisé, de sorte que vous n'avez pas à spécifier toutes propriétés d'un objet lorsque vous créez un profil - seulement celles que vous voulez modifier. Par défaut, le profil maître est nommé "**Master**". Vous pouvez changer le nom du profil maître dans le volet **Property Profiles** dans la fenêtre Préférences.

7.7.3 Bascule entre profils

Basculer un seul objet

Pour changer le profil d'un objet, vous pouvez utiliser l'inspecteur de propriété de l'objet ou la propriété **revProfile**.

```
set the revProfile of player "My Player" to "MyProfile"
```

Basculer tous les objets d'une carte

Pour changer les profils de tous les objets sur une carte, utiliser la commande **revSetCardProfile** :

```
revSetCardProfile "MyProfile","My Stack"
```

La déclaration ci-dessus définit le profil de tous les objets la carte actuelle de la pile nommé "My Stack". (Bien que la commande **revSetCardProfile** change une carte, vous indiquez un nom pile, pas un nom de carte.) Si un objet sur la carte ne possède pas de profil avec le nom spécifié, l'objet est laissé intact.

Basculer tous les objets d'une pile

Pour changer les profils de tous les objets dans une pile, utilisez la commande **revSetStackProfile** :

```
revSetStackProfile "MyProfile","My Stack"
```

La déclaration ci-dessus définit le profil de tous les objets dans la pile nommée "My Stack". Si un objet dans la pile n'a pas de profil avec le nom spécifié, l'objet est laissé intact.

Basculer tous les objets d'un fichier pile

Pour changer les profils de tous les objets dans chaque pile dans un fichier de pile, utilisez la commande revSetStackFileProfile :

```
revSetStackFileProfile "MyProfile","My Stack"
```

La déclaration ci-dessus définit le profil de tous les objets dans la pile nommée "My Stack", ainsi que toutes autres piles dans le même fichier de la pile.
Si un objet dans l'une des piles n'a pas de profil avec le nom spécifié, l'objet est laissé intact.

7.7.4 Création d'un profil dans un gestionnaire

En plus de créer des profils de propriétés dans l'inspecteur de propriétés, vous pouvez créer un profil dans un gestionnaire.
Pour permettre la création de profils, cochez la case "***Create profiles automatically***" dans le volet "***Property Profiles***", de la fenêtre ***Preferences***. Si cette case est cochée, en définissant la propriété revProfile de l'objet on crée automatiquement le profil.
Cette capacité est particulièrement utile si vous voulez créer un certain nombre de profils, comme le montre l'exemple suivant :

```
on mouseUp
  -- crée un profil pour chaque carte dans la pile
  repeat with thisCard = 1 to the number of cards
    set the revProfile of card x to "monNouveauProfil"
  end repeat
end mouseUp
```

Le gestionnaire ci-dessus crée un profil appelé "monNouveauProfil" pour toutes les cartes dans la pile actuelle.

Dans l'ordre, pour que ce gestionnaire fonctionne, l'option "***Create profiles automatically***" dans le volet "***Property Profiles***" de la fenêtre des ***Preferences*** doit être activée.

Vous pouvez contrôler ce comportement soit dans la fenêtre ***Preferences*** ou en utilisant le mot-clé **gRevProfileReadOnly**.
Si vous ne voulez pas enregistrer les modifications de propriété lors du basculement des profils, effectuez l'une des actions suivantes:

Définissez la variable, **gRevProfileReadOnly**, sur vrai :

```
global gRevProfileReadOnly
put true into gRevProfileReadOnly
```

Dans le volet ***Property Profiles***, de la fenêtre ***Preferences***, décochez la case **Don't save changes in profile**. Les deux méthodes de modification de ce paramètre sont équivalentes : changer la variable **gRevProfileReadOnly** modifie également le paramètre de préférence, et vice versa.

Pour plus de détails, voir **gRevProfileReadOnly** dans le dictionnaire de LiveCode.

7.7.5 Ajout de paramètres de profil dans un gestionnaire

Vous pouvez ajouter un paramètre de propriété sur un profil en basculant le profil, puis en définissant la propriété :

```
set the revProfile of button 1 to "MyProfile"
set the foregroundColor of button 1 to "red"
set the revProfile of button 1 to "Master"
```

Par défaut, si vous modifiez une propriété et ensuite basculez les profils, les propriétés modifiées et leurs paprmètres actuels sont enregistrés avec le profil.

7.8 Propriétés personnalisées

Une propriété personnalisée est une propriété que vous créez pour un objet, en plus de ses propriétés intégrées. Vous pouvez définir des propriétés personnalisées pour n'importe quel objet, et les utiliser pour stocker tout type de données.
Cette rubrique explique comment créer et utiliser des propriétés personnalisées, et comment organiser des propriétés personnalisées dans des ensembles (ou tableaux). La section suivante explique comment créer des propriétés virtuelles et utiliser les gestionnaires **getProp** et **setProp** pour gérer les demandes de propriétés personnalisées.

7.8.1 Utilisation des propriétés personnalisées

Une propriété personnalisée est une propriété que vous définissez. Vous pouvez créer autant de propriétés personnalisées pour un objet que vous voulez, et de mettre tout type de données en elles (même les données binaires). Vous pouvez même stocker un fichier dans une propriété personnalisée.
Utilisez une propriété personnalisée lorsque vous voulez :

- des données associées à un objet spécifique
- enregistrer les données avec l'objet dans le fichier de pile
- accéder aux données rapidement

7.8.2 Création d'une propriété personnalisée

Vous créez une propriété personnalisée en définissant la nouvelle propriété avec une valeur. Si vous définissez une propriété personnalisée qui n'existe pas, LiveCode crée automatiquement la propriété personnalisée et lui affecte la valeur demandée.
Cela signifie que vous pouvez créer une propriété personnalisée dans un gestionnaire ou la boîte de message, simplement en utilisant la commande set. L'instruction suivante crée une propriété personnalisée appelée "endingTime" pour un bouton :

```
set the endingTime of button "Session" to the long time
```

Vous pouvez créer des propriétés personnalisées pour n'importe quel objet. Cependant, vous ne pouvez pas créer des propriétés personnalisées globales, ou des propriétés personnalisées pour un bloc de texte dans un champ. Contrairement à certaines propriétés intégrées, une propriété personnalisée s'applique uniquement à un objet.
Important : Chaque objet peut avoir ses propres propriétés personnalisées et les propriétés personnalisées

ne sont pas partagées entre les objets. La création d'une propriété personnalisée pour un objet n'en crée pas pour d'autres objets.

7.8.3 Le contenu d'une propriété personnalisée

Vous définissez la valeur d'une propriété personnalisée en utilisant son nom de propriété avec la commande **set**, de la même façon que vous définissez les propriétés intégrées :

```
set the myCustomProperty of button 1 to false
```

Vous pouvez voir et modifier toutes les propriétés personnalisées d'un objet dans le volet de l'inspecteur de propriétés de l'objet Propriétés personnalisées : cliquez sur la propriété personnalisée que vous souhaitez modifier, puis entrez la nouvelle valeur.

Modifier une partie d'une propriété

Comme les propriétés intégrées, les propriétés personnalisées ne sont pas des conteneurs, de sorte que vous ne pouvez pas utiliser une expression de bloc pour changer une partie de la propriété personnalisée. Au lieu de cela, vous placez la valeur de la propriété dans une variable et modifiez celle-ci, puis définissez la propriété personnalisée en retour avec le nouveau contenu de la variable :

```
put the lastCall of this card into myVar -- copie propriété lastCall dans variable myVar
put "March" into word 3 of myVar -- modifie un bloc (word 3) dans la variable myVar
set the lastCall of thisCard to myVar --modifie lastCall depuis variable myVar
```

7.8.4 Noms de propriété personnalisée

Le nom d'une propriété personnalisée doit être composé d'un seul mot et peut contenir n'importe quelle combinaison de lettres, chiffres et caractère de soulignement _ . Le premier caractère doit être une lettre ou un trait de soulignement.
Évitez de donner à une propriété personnalisée le même nom qu'une variable. Si vous vous référez à une propriété personnalisée dans un gestionnaire, et qu'il y ait une variable du même nom, LiveCode utilise le contenu de la variable comme le nom de la propriété personnalisée. Cela provoque souvent des résultats inattendus.

Important : Les noms de propriétés personnalisées commençant par **rev** sont réservés aux propriétés personnalisées de LiveCode. Nommer une propriété personnalisée avec un nom réservé peut produire des résultats inattendus lorsque vous travaillez dans l'environnement de développement.

7.8.5 Se référer à des propriétés personnalisées

Les références aux propriétés personnalisées ressemblent à s'y méprendre aux références des propriétés intégrées : le mot **the**, le nom de propriété, le mot **of**, et une référence à l'objet.
Par exemple, pour utiliser une propriété personnalisée appelée "LastCall" qui appartient à une carte, utiliser une déclaration comme suit :

```
put the lastCall of this card into field "Date"
```

Comme les propriétés intégrées, les propriétés personnalisées sont sources de valeur, de sorte que vous pouvez obtenir la valeur d'une propriété personnalisée à l'aide d'une expression. La valeur de la propriété est substituée à la référence de propriété lorsque l'instruction est exécutée.
Par exemple, si "LastCall" propriété personnalisée de la carte, est "Today", l'exemple énoncé ci-dessus met la chaîne "Today" dans le champ "Date".

7.8.6 Propriétés personnalisées inexistantes

Les propriétés personnalisées qui n'existent pas, sont évaluées comme vides. Par exemple, si la carte actuelle n'a pas une propriété personnalisée appelée **LastCall**, la déclaration suivante vide le champ :

```
put the lastCall of this card into field "Date" -- vide le champ "Date"
```

Note : Se référer à une propriété personnalisée inexistante ne provoque pas une erreur de script. Cela signifie que si vous avez mal orthographié le nom d'une propriété personnalisée dans un gestionnaire, vous n'obtiendrez pas de message d'erreur, alors vous pourriez ne pas remarquer le problème tout de suite.

7.8.7 Savoir si une propriété personnalisée existe

La propriété **customKeys** d'un objet répertorie les propriétés personnalisées de l'objet, ligne par ligne:

```
put the customKeys of button 1 into field "Custom Props"
```

Pour savoir si une propriété personnalisée pour un objet existe, vous vérifiez si elle est présente dans le **customKeys** de l'objet.
La déclaration suivante vérifie si un lecteur possède une propriété personnalisée appelée "doTellAll" :

```
if "doTellAll" is among the lines of the customKeys of player "My Player" then...
```

Vous pouvez également chercher dans le volet Propriétés personnalisées, de l'inspecteur de propriétés de l'objet, qui dresse la liste des propriétés personnalisées.
Voir le chapitre Création d'une interface utilisateur pour plus de détails.

7.8.8 Propriétés personnalisées et Conversion de texte entre plateformes

Lorsque vous déplacez une pile développée sur un système Mac OS X ou un OS sur système Windows ou Unix (ou vice versa), LiveCode traduit automatiquement le texte dans les champs et les scripts dans le jeu de caractères correspondant. Toutefois, le texte des propriétés personnalisées n'est pas converti entre l'ISO et les jeux de caractères Macintosh. C'est parce que les propriétés personnalisées peuvent contenir des données binaires ainsi que du texte, et en les convertissant, les données seraient brouillées.
Les caractères ASCII dont la valeur se situe entre 128 et 255, comme les cotes incurvées et des caractères accentués, n'ont pas la même valeur ASCII dans le jeu de caractères Mac OS et le caractère ISO 8859-1 ensemble utilisé sur les systèmes Unix et Windows. Si un tel caractère est dans un champ, il est automatiquement traduit, mais il n'est pas traduit si elle est dans une propriété personnalisée.

Pour cette raison, si votre pile présente des propriétés personnalisées à l'utilisateur - par exemple, si la pile copie une propriété personnalisée dans un champ - et si le texte contient des caractères spéciaux, il peut ne pas s'afficher correctement si vous déplacez la pile entre les plates-formes. Pour éviter ce problème, appliquez l'une des méthodes suivantes :

Avant d'afficher la propriété personnalisée, convertir le jeu de caractères approprié à l'aide de la fonction macToISO ou ISOToMac. L'exemple suivant montre comment convertir une propriété personnalisée qui a été créé sur un système Mac OS, lorsque la propriété est affichée sur un système Unix ou Windows :

```
if the platform is "MacOS" then
  answer the myPrompt of button 1
else
  answer macToISO (the myPrompt of button 1)
end if
```

Au lieu de stocker la propriété personnalisée sous forme de texte, l'enregistrer comme **HTML**, en utilisant la propriété **HTMLText** des champs :

```
set the myProp of this card to the HTMLText of field 1 -- stocker le contenu HTML du champ 1 dans la myProp'""
```

Parce que la propriété **HTMLText** encode les caractères spéciaux comme des entités, il s'assure que la propriété personnalisée ne contienne pas de caractères spéciaux.
Vous pouvez alors placer le HTMLText du champ, comme contenu de la propriété personnalisée pour l'afficher :

```
set the HTMLText of field "Display" to the myProp of this card -- récupérer le contenu de myProp dans le champ Display
```

7.8.9 Stocker un fichier dans une propriété personnalisée

Vous pouvez utiliser une **URL** pour stocker le contenu d'un fichier dans une propriété personnalisée :

```
set the myStoredFile of stack "My Stack" to URL "binfile:mypicture.jpg"
```

Vous restaurez le fichier en mettant la valeur de la propriété personnalisée dans une **URL** :

```
put the myStoredFile of stack "My Stack" into URL "binfile:mypicture.jpg"
```

Parce qu'une propriété personnalisée peut contenir tout type de donnée, vous pouvez stocker soit des fichiers texte soit des fichiers binaires dans une propriété personnalisée. Vous pouvez utiliser cette fonctionnalité pour regrouper les fichiers multimédia ou d'autres fichiers dans votre pile.

Astuce : Pour économiser de l'espace, compresser le fichier avant de le stocker :

```
set the myStoredFile of stack "My Stack" to compress(URL "binfile:mypicture.jpg")
```

Lors de la restauration du fichier, décompressez-le d'abord:

```
put decompress(the myStoredFile of stack "My Stack") into URL "binfile:mypicture.jpg"
```

Pour plus d'informations sur l'utilisation des conteneurs d'**URL**, voir le chapitre Travailler avec les fichiers, les **URL** et les Sockets.

7.8.10 Suppression d'une propriété personnalisée

Comme décrit ci-dessus, la propriété **customKeys** d'un objet est une liste de propriétés personnalisées de l'objet. Vous pouvez définir les **customKeys** d'un objet pour contrôler les propriétés personnalisées dont il dispose.
Dans LiveCode, il n'y a pas de commande pour supprimer une propriété personnalisée .
A la place, vous copiez tous les noms de propriétés personnalisées dans une variable, supprimez celui dont vous ne voulez plus dans cette variable, et redéfinissez les **customKeys** de l'objet en retour depuis les contenus modifiés de la variable. Cela supprime la propriété personnalisée dont le nom a été supprimé. Par exemple, les énoncés suivants suppriment une propriété personnalisée appelée "***propertyToRemove***" à partir du bouton "***My Button***":

```
get the customKeys of button "My Button"
set the wholeMatches to true
delete line lineOffset("propertyToRemove",it) of it
set the customKeys of button "My Button" to it
```

Vous pouvez également supprimer une propriété personnalisée dans le volet de l'inspecteur des propriétés de l'objet Propriétés personnalisées. Sélectionnez le nom de la propriété et cliquez sur le bouton ***Delete*** pour le supprimer.

7.9 Jeux de propriétés personnalisées

Les propriétés personnalisées peuvent être organisées en ensembles (jeux) de propriétés personnalisées - ou en tableaux de propriétés personnalisées. Un jeu de propriétés personnalisées est un groupe de propriétés personnalisées qui a un nom que vous spécifiez.
Quand vous vous référez à une propriété personnalisée, LiveCode regarde pour cette propriété, dans le jeu de propriétés personnalisées actuellement actif de l'objet. Lorsque vous créez ou définissez une propriété personnalisée, LiveCode la crée dans l'ensemble des propriétés personnalisées actuellement actif, ou définit la valeur de cette propriété dans l'ensemble actuellement actif. Un jeu de propriétés personnalisées est actif à un moment, mais vous pouvez utiliser une sortie de type table pour obtenir ou définir des propriétés personnalisées dans des ensembles autres que le jeu actuel.
Les exemples de la section précédente supposent que vous n'avez pas créé de jeux de propriétés personnalisées. Si vous créez une propriété personnalisée sans créer une propriété personnalisée prévue pour cela, comme le montrent les exemples précédents, la nouvelle propriété personnalisée devient partie intégrante par défaut du jeu de propriété personnalisée de l'objet.

7.9.1 Création de jeux de propriétés personnalisées

Pour rendre un jeu de propriétés personnalisées actif, vous définissez la propriété **customPropertySet** de l'objet pour le jeu que vous souhaitez utiliser. Comme pour les propriétés personnalisées et les variables locales, si la propriété personnalisée définie que vous spécifiez n'existe pas, LiveCode la crée automatiquement, ainsi vous pouvez créer un jeu de propriétés personnalisées pour un objet simplement en basculant sur cet ensemble.
L'instruction suivante crée un ensemble de propriétés personnalisées appelé "Alternate" pour un objet, et rend le jeu actif :

```
set the customPropertySet of the target to "Alternate"
```

La déclaration ci-dessus crée l'ensemble de la propriété personnalisée.

Vous pouvez également afficher, créer et supprimer des ensembles de propriétés personnalisées dans le volet **Custom** de l'inspecteur des propriétés de l'objet.
Vous pouvez répertorier tous les ensembles de propriétés personnalisées d'un objet en utilisant sa propriété **customPropertySets**.

Comme pour les propriétés personnalisées, vous pouvez créer des ensembles de propriétés personnalisées pour n'importe quel objet. Mais vous ne pouvez pas créer des ensembles globaux de propriétés personnalisées, ou des ensembles de propriétés personnalisées pour un bloc dans un champ.

7.9.2 Les noms de jeux de propriétés personnalisées

Les noms des ensembles de propriétés personnalisées devraient consister en un seul mot, avec n'importe quelle combinaison de lettres, chiffres et caractères de soulignement (_). Le premier caractère doit être une lettre ou un trait de soulignement.
Il est possible de créer un ensemble de propriétés personnalisées avec un nom qui a plus d'un mot, ou qui n'est pas conforme à ces directives. Cependant, ce n'est pas recommandé, car un tel jeu de propriétés personnalisées ne peut pas être utilisé avec la notation de type tableau décrit ci-dessous.

Note : Lorsque vous utilisez le volet **CustomProperties** dans l'inspecteur de propriétés pour créer un jeu de propriétés personnalisées, le volet vous limite à ces lignes directrices.

7.9.3 Se référer à des jeux de propriétés personnalisées

Pour changer le jeu de propriétés personnalisées actif, définissez la propriété **customPropertySet** de l'objet pour le nom du jeu que vous souhaitez utiliser :

```
set the customPropertySet of button 3 to "Spanish"
```

Toute référence à des propriétés personnalisées renvoient au jeu des propriétés personnalisées actuel. Par exemple, supposons que vous avez deux propriétés personnalisées définie nommées **Espagnol** et **Français**, et le jeu **Français** comporte une propriété personnalisée appelée **Paris**, tandis que le jeu **Espagnol** n'en a pas. Si vous basculez vers le jeu **Espagnol**, les **customKeys** de l'objet ne comprennent pas **Paris**, parce que le jeu de propriétés personnalisées actuel n'inclut pas cette propriété.

Si vous vous référez à une propriété personnalisée qui n'est pas dans le jeu actuel, la référence est évaluée comme vide. Si vous définissez une propriété personnalisée qui n'est pas dans la configuration actuelle, la propriété personnalisée est créée dans l'ensemble. Vous pouvez avoir deux propriétés personnalisées avec le même nom dans différents ensembles de propriétés personnalisées, mais ils ne s'affectent pas l'un l'autre : changer l'un, ne modifie pas l'autre.

La propriété **customProperties** d'un objet inclut uniquement les propriétés personnalisées qui sont dans le jeu de propriétés personnalisées en cours.
Pour spécifier les **customProperties** d'un jeu de propriétés personnalisées particulier, vous devez inclure le nom de l'ensemble entre crochets :

```
put the customProperties[mySet] of this card into myArray
```

7.9.4 Savoir si un ensemble de propriétés personnalisées existe

La propriété **customPropertySets** d'un objet liste des jeux de propriétés personnalisées de l'objet, ligne par ligne :

```
answer the customPropertySets of field "My Field"
```

Pour savoir si un jeu de propriétés personnalisées pour un objet existe, vous vérifiez si il est présent dans le **customPropertySets** de l'objet. La déclaration suivante, vérifie si une image a un ensemble de propriétés personnalisées appelé "Spanish":

```
if "Spanish" is among the lines of the customPropertySets of image ID 23945 then...
```

Vous pouvez également chercher dans le volet de l'inspecteur de propriétés de l'objet, qui répertorie les jeux de propriétés personnalisées dans le menu **Set** à mi-chemin au bas de la fenêtre des propriétés personnalisées.

7.9.5 Le set de propriété personnalisée par défaut

Par défaut le jeu de propriété personnalisée d'un objet est l'ensemble qui est actif si vous n'avez pas utilisé la propriété **customPropertySet** pour basculer sur un autre jeu. Chaque objet possède un jeu de propriétés personnalisé par défaut, vous n'avez pas besoin de le créer.

Si vous créez une propriété personnalisée sans d'abord passer à un jeu de propriétés personnalisé - comme dans les exemples précédents sur ce sujet - la propriété personnalisée est créée dans le jeu par défaut. Si vous ne définissez pas la propriété **customPropertySet**, tous vos propriétés personnalisées sont créées dans le jeu par défaut.

Le set de propriété personnalisée par défaut n'a pas de nom propre, et n'est pas répertorié dans la propriété **customPropertySets** de l'objet. Pour basculer depuis un autre jeu à l'ensemble par défaut, vous définissez **customPropertySet** de l'objet à vide :

```
set the customPropertySet of the target to empty
```

7.9.6 L'utilisation de plusieurs sets de propriétés personnalisées

Comme un seul jeu de propriétés personnalisées peut être actif à la fois, vous pouvez créer des propriétés personnalisées distinctes avec le même nom mais des valeurs différentes sur différents ensembles. La valeur que vous obtenez dépend du jeu de propriétés personnalisées qui est actuellement actif.

Un exemple de traduction

Supposons que votre pile utilisant plusieurs propriétés personnalisées qui détiennent des chaînes en anglais, doive être affichée à l'utilisateur par différentes commandes. Votre pile peut contenir un énoncé tel que ceci :

```
answer the standardErrorPrompt of this stack
```

La déclaration ci-dessus affiche le contenu de la propriété personnalisée appelée **standardErrorPrompt** dans une boîte de dialogue.
Supposons que vous décidiez de traduire votre demande en français. Pour ce faire, vous passez votre jeu original de propriétés personnalisées anglais dans un jeu de propriétés personnalisées (que vous pourriez appeler **myEnglishStrings**), et vous créez un nouvel ensemble appelé **myFrenchStrings** pour avoir les propriétés traduites.
Chaque jeu possède les propriétés de même nom, mais les valeurs dans un jeu sont en français et dans l'autre en anglais. Vous basculez entre les jeux en fonction du langage que l'utilisateur choisit. La déclaration :

```
answer the standardErrorPrompt of this stack
```

propose anglais ou français, selon le jeu de propriétés personnalisées actif : "myEnglishStrings" ou "myFrenchStrings".

7.9.7 Copie des propriétés personnalisées entre des jeux de propriétés

Quand il est créé, un jeu de propriétés personnalisées est vide, c'est à dire qu'il n'y a pas de propriétés personnalisées en lui. Vous mettez des propriétés personnalisées dans un nouveau jeu de propriété personnalisée, en créant les propriétés personnalisées alors que le jeu est actif :

```
set the customPropertySet of button 1 to "MyNewSet"  -- un nouveau jeu de propriétés personnalisées rendu actif
set the myCustomProp of button 1 to true             -- une nouvelle propriété personnalisée dans le jeu actif
```

Vous pouvez également utiliser la propriété **customProperties** (examinée précédemment dans cette rubrique) pour copier les propriétés personnalisées entre les ensembles. Par exemple, supposons que vous avez créé un ensemble complet de propriétés personnalisées regroupées dans un jeu de propriété personnalisée appelée **myEnglishStrings**, et que vous vouliez les copier sur un nouvel ensemble de propriétés personnalisées, **myFrenchStrings**, de sorte pouvoir facilement les traduire. Les instructions suivantes créent le nouveau jeu de propriété personnalisée, puis copient toutes les propriétés de l'ancien ensemble vers la nouvelle :

```
-- création du nouveau jeu de propriétés personnalisées
set the customPropertySet of this stack to "myFrenchStrings"
-- copie des propriétés personnalisées du jeu English vers le nouveau French
set the customProperties ["myFrenchStrings"] of this stack to the customProperties ["myEnglishStrings"] of this stack
```

Attention : Jeux de propriétés personnalisées dans l'environnement de développement
L'environnement de développement de LiveCode utilise les propriétés personnalisées pour créer une grande partie de sa propre interface utilisateur. Toutes ces propriétés sont stockées dans des ensembles de propriétés personnalisées dont le nom commence par "**CREV**". Si vous créez des propriétés personnalisées, mais n'envisagez pas de les organiser dans vos propres ensembles de propriétés personnalisées, vous n'avez pas à vous soucier des propriétés personnalisées de LiveCode. Sauf si vous modifiez la **customPropertySet** d'un objet, vos propres gestionnaires ne pourront jamais interagir avec eux.
Toutefois, si vous utilisez des jeux de propriétés personnalisées - par exemple, si vous avez une boucle de répétition qui passe en revue tous les jeux de propriétés personnalisées d'un objet - assurez-vous d' ignorer tous ceux dont les noms commencent par "**CREV**". Cela garantit que vous n' interférerez pas accidentellement avec une des propriétés personnalisées réservées de LiveCode.

7.9.8 tables (Array), propriétés personnalisées et ensembles de propriétés personnalisées

Toutes les propriétés personnalisées dans un jeu de propriétés personnalisées forment un réseau. Le nom du tableau est le nom du jeu de propriétés personnalisées et les éléments du tableau sont les propriétés personnalisées individuelles dans ce jeu de propriétés personnalisées.

Se référer aux propriétés personnalisées en utilisant notation de typetableau (array)

Vous pouvez utiliser la sortie de type table pour pointer les propriétés personnalisées dans n'importe quel jeu de propriétés personnalisées. Cela vous permet, d'obtenir et de définir, n'importe quelle propriété personnalisée, même si elle n'est pas dans le set en cours, sans changer ce dernier.

Par exemple, supposons qu'un bouton ait une propriété personnalisée appelée **myprop**, qui est dans un jeu de propriétés personnalisées appelé **mySet**. Si **mySet** est le set en cours, vous pouvez vous référer à la propriété **myprop** , comme ceci:

```
get the myProp of button 1
set the myProp of the target to 20
```

Mais vous pouvez également utiliser notation de type tableau pour faire référence à la propriété **myprop**, même si **mySet** n'est pas le set en cours. Pour faire référence à cette propriété personnalisée quel que soit le jeu de propriétés personnalisées actif utilisez des déclarations telles que les suivantes :

```
get the mySet["myProp"] of button 1
set the mySet["myProp"] of the target to 20
```

Note : Parce que le set de propriétés personnalisée par défaut n'a pas de nom, vous ne pouvez pas utiliser la notation de type **array** pour désigner une propriété personnalisée dans le jeu par défaut.

Stocker untableau dans un jeu de propriétés personnalisées

Si vous stockez une série de propriétés personnalisées dans un jeu de propriétés personnalisées, l'ensemble peut être utilisé comme un tableau.
Vous pouvez penser le jeu de propriétés personnalisées, comme s'il s'agissait d'une seule propriété personnalisée, et les propriétés de l'ensemble en tant qu'éléments individuels du tableau.

Pour stocker une variable de tableau comme un ensemble de propriétés personnalisées, utiliser une déclaration comme suit :

```
set the customProperties["myProperty"] of me to theArray
```

La déclaration ci-dessus crée un jeu de propriétés personnalisées, appelé **myProperty**, et stocke chaque élément, dans ***theArray***, comme une propriété personnalisée dans le nouvel ensemble.

Pour récupérer un seul élément du tableau, utilisez une déclaration comme ceci :

```
get the myProperty["myElement"] of field "Example"
```

7.9.9 Suppression d'un jeu de propriétés personnalisées

Comme décrit ci-dessus, la propriété **customPropertySets** d'un objet est une liste de jeux de propriétés personnalisées de l'objet. Vous pouvez définir les **customPropertySets** d'un objet, pour contrôler les jeux de propriétés personnalisées, dont il dispose.

Avec LiveCode, il n'y a pas de commande pour supprimer un ensemble de propriétés personnalisées. Au lieu de cela, vous placez tous les noms de jeux de propriétés personnalisées dans une variable, supprimez celui que vous ne voulez pas dans cette variable, et réglez en retour les **customPropertySets** depuis le contenu modifié de la variable. Cela supprime le jeu de propriétés personnalisées qui a le nom que vous avez supprimé.
Par exemple, les énoncés suivants suppriment un ensemble de propriétés personnalisées, appelé **mySet**, du bouton **My Button** :

```
get the customPropertySets of button "My Button"
set the wholeMatches to true
delete line lineOffset("mySet",it) of it
set the customPropertySets of button "My Button" to it
```

Vous pouvez également supprimer un ensemble de propriétés personnalisées dans le volet ***Custom Properties*** de l'inspecteur de propriétés de l'objet. Sélectionnez le nom du jeu dans le menu ***Set***, puis cliquez sur le bouton ***Delete*** pour le supprimer.

7.10 Attacher des gestionnaires à des propriétés personnalisées

Lorsque vous modifiez une propriété personnalisée, LiveCode envoie un déclencheur **setProp** à l'objet dont la propriété est changée. Vous pouvez écrire un gestionnaire **setProp** pour piéger ce déclencheur et répondre à la tentative de changer la propriété. Tel un message, ce déclencheur utilise le chemin de message, de sorte que vous pouvez placer le gestionnaire **setProp** n'importe où dans le chemin du message de l'objet. De même, quand vous obtenez la valeur d'une propriété personnalisée, LiveCode envoie un appel **getProp** à l'objet dont la propriété est interrogée. Vous pouvez écrire un gestionnaire **getProp** pour répondre à la demande de renseignements. Comme un appel de fonction, l'appel **getProp**, traverse également le chemin d'un message.

En utilisant **getProp** et **setProp** comme gestionnaires, vous pouvez :

- valider la valeur d'une propriété personnalisée avant de l'y placer
- rapporter la valeur d'une propriété personnalisée dans un format différent que celui qui est stocké
- assurer l'intégrité d'une collection de propriétés en les éditant toutes en même temps
- changer le comportement d'un objet quand une propriété personnalisée est modifiée

Note : Les déclencheurs **setProp** et les appels **getProp** ne sont pas envoyés quand une propriété intégrée est modifiée ou accédée. Ils ne s'adressent qu'aux propriétés personnalisées.

7.10.1 Répondre à l'évolution d'une propriété personnalisée

Lorsque vous utilisez la commande **set** pour modifier une propriété personnalisée, LiveCode envoie un déclencheur **setProp** à l'objet dont la propriété est changée.

Un déclencheur de **setProp** agit un peu comme le fait un message. Il est envoyé à un objet particulier. Si le script de cet objet contient un gestionnaire **setProp** pour la propriété, le gestionnaire est exécuté, sinon, le déclencheur se déplace le long du chemin du message jusqu'à ce qu'il trouve un gestionnaire pour la propriété. S'il arrive à la fin du trajet de message sans être piégé, le déclencheur **setProp** définit la propriété personnalisée à sa nouvelle valeur. Pour plus d'informations sur le chemin du message, voir la section sur le chemin du message.
Vous pouvez inclure autant de gestionnaires **setProp** dans un script pour autant de propriétés personnalisées que vous le désirez.

7.10.2 La structure d'un gestionnaire setProp

Contrairement à un gestionnaire de messages, un gestionnaire de **setProp** commence par le mot **setProp** à la place du mot **on**. Il est suivi par le nom du gestionnaire (qui est le même que le nom de la propriété personnalisée) et d'un paramètre qui contient la nouvelle valeur de la propriété. Un gestionnaire de **setProp**, comme tous les gestionnaires, se termine par le mot **end** suivi par le nom du gestionnaire. L'exemple suivant montre un gestionnaire **setProp**, pour une propriété personnalisée, nommé **percentUsed**, et qui peut être placé dans le script de l'objet dont la propriété personnalisée est :

```
setProp percentUsed newAmount -- répond à : définir la propriété percentUsed
  if newAmount is not a number or newAmount < zero or newAmount > 100 then
   beep 2
   exit percentUsed
  end if
  pass percentUsed
end percentUsed
```

Lorsque vous définissez la propriété personnalisée **percentUsed**, le gestionnaire **percentUsed** est exécuté :

```
set the percentUsed of scrollbar "Progress" to 90
```

Lorsque cette instruction est exécutée, LiveCode envoie un déclencheur **setProp** à la barre de défilement. La nouvelle valeur de 90 est placée dans le paramètre **newAmount**. Le gestionnaire fait en sorte que la nouvelle valeur est dans la gamme 0-100; sinon, il émet un signal sonore et quitte le gestionnaire, empêchant la propriété d'être réglée.

Pour plus de détails sur la structure de contrôle de **setProp**, voir **setProp** dans le Dictionnaire de LiveCode.

Passer le déclenchement de setProp

Lorsque déclencheur **setProp** atteint le moteur - la dernière étape du trajet pour le message - la propriété personnalisée est activée. Si le déclencheur est piégé et ne parvient pas au moteur, la propriété personnalisée n'est pas définie.
Pour laisser un déclencheur passer plus loin sur le chemin du message, utilisez la structure de contrôle **pass**. La structure de contrôle **pass** arrête le gestionnaire en cours et envoie le déclencheur à l'objet suivant dans le chemin de message, comme si l'objet n'avait pas de gestionnaire pour la propriété personnalisée.

Dans le gestionnaire **percentUsed** ci-dessus, si **newAmount** est hors de portée, le gestionnaire utilise la structure de contrôle sortie pour l'arrêter, sinon il exécute la structure de contrôle **pass**. Si la valeur **newAmount** est dans la bonne gamme, la structure de contrôle **pass** permet à la propriété d'être définie. Sinon, puisque le déclenchement n'est pas passé, il n'atteint jamais le moteur, donc la propriété n'est pas modifiée.
Vous pouvez utiliser cette fonctionnalité pour vérifier la valeur d'une propriété personnalisée avant de l'autoriser à être définie. Par exemple, si une propriété personnalisée est censée être booléenne (vrai ou faux), un gestionnaire de **setProp** peut piéger le déclenchement si la valeur est tout sauf vrai ou faux :

```
setProp myBoolean newValue
  if newValue is true or newValue is false
  then pass myBoolean
exit myBoolean
```

Utilisation du chemin de message avec un déclencheur setProp

Comme les déclencheurs **setProp** utilisent le chemin de message, un seul objet peut recevoir les déclencheurs **setProp**, pour tous les sous-objets qui lui appartiennent. Par exemple, les déclencheurs **setProp**, pour tous les contrôles sur la carte sont envoyés à la carte, si le script de commande n'a pas de gestionnaire pour cette propriété. Vous pouvez profiter du chemin de message pour mettre en œuvre le même comportement **setProp** pour les objets qui ont tous la même propriété personnalisée.

Attention : Si un gestionnaire **setProp** définit sa propriété personnalisée pour un objet qui possède ce gestionnaire **setProp** dans son chemin du message, un emballement récursif se produira. Pour éviter ce problème, définissez la propriété **lockMessages** sur vrai avant de définir la propriété personnalisée. Par exemple, supposons que toutes les cartes dans votre pile ont une propriété personnalisée appelée "LastChanged". Au lieu de mettre un gestionnaire **setProp** de cette propriété dans le script de chaque carte, vous pouvez placer un seul gestionnaire dans le script de la pile :

```
setProp lastChanged newDate
   convert newDate to seconds
   lock messages -- éviter le phénomène de récursion
   set the lastChanged of the target to newDate
   unlock messages
end lastChanged
```

Note: Pour faire référence à l'objet dont la propriété est définie, utilisez la fonction **target**. **target** réfère à l'objet qui a reçu le premier le déclencheur **setProp** (l'objet dont la propriété personnalisée est en cours de définition) même si le gestionnaire est en cours d'exécution dans le script d'un autre objet.

Définition des propriétés dans un gestionnaire setProp

Dans l'exemple ***LastChanged*** ci-dessus, le gestionnaire définit la propriété personnalisée directement, au lieu de simplement passer le déclenchement de **setProp**. Vous devez utiliser cette méthode si le gestionnaire apporte une modification à la valeur de la propriété, parce que la structure de contrôle **pass** transmet simplement la valeur initiale de la propriété.

Important : Si vous utilisez la commande **set** dans un gestionnaire **setProp** pour définir la même propriété personnalisée pour l'objet courant, aucun déclencheur **setProp** n'est envoyé à l'objet cible. (Il s'agit d'éviter l'emballement récursif, lorsque le gestionnaire **setProp** se déclenche lui-même.) La définition d'une propriété personnalisée différente envoie un déclencheur **setProp**. Ainsi en est-il pour définir la propriété personnalisée du gestionnaire d'un un objet autre que celui-ci, dont le scénario contient le gestionnaire **setProp**.

En utilisant cette méthode, vous pouvez non seulement vérifier la valeur d'une propriété, et lui permettre d'être modifiée seulement si elle est dans la gamme, vous pouvez également modifier la valeur afin qu'elle soit dans la plage correcte, qu'elle ait le bon format et ainsi de suite .
L'exemple suivant est similaire au gestionnaire **percentUsed** ci-dessus, mais au lieu d'un signal sonore si le ***newAmount*** est hors de portée, il force la nouvelle valeur dans la gamme 0-100 :

```
setProp percentUsed newAmount
  set the percentUsed of the target to max(zero,min(100,newAmount))
end percentUsed
```

Propriétés inexistantes

Si la propriété personnalisée spécifiée par un gestionnaire **setProp** n'existe pas, le gestionnaire **setProp** est toujours exécuté quand un gestionnaire définit la propriété. Si le gestionnaire transmet le déclencheur **setProp**, la propriété personnalisée est créée.

Ensembles de propriétés personnalisées et les gestionnaires setProp

Un gestionnaire de **setProp** pour un ensemble de propriétés personnalisées se comporte différemment d'un gestionnaire **setProp** pour une propriété personnalisée qui est dans le jeu par défaut.

Lorsque vous définissez une propriété personnalisée dans un jeu de propriétés personnalisées, le déclencheur **setProp** est nommé pour l'ensemble, pas la propriété. Le nom de la propriété est passée à un paramètre en utilisant une notation spéciale. Cela signifie que, pour les propriétés personnalisées dans un jeu, vous écrivez un gestionnaire **setProp** unique pour le jeu, plutôt que pour chaque propriété individuelle.

L'exemple suivant traite des déclencheurs de **setProp** pour toutes les propriétés personnalisées dans un jeu de propriétés personnalisées, appelé ***myFrenchStrings***, qui contient des propriétés personnalisées nommées ***standardErrorPrompt***, ***filePrompt***, et peut-être d'autres propriétés personnalisées :

```
setProp myFrenchStrings[myPropertyName] newValue
  -- Le paramètre myPropertyName contient le nom de la propriété en cours de réglage
  switch myPropertyName
  case "standardErrorPrompt"
    set the myFrenchStrings["standardErrorPrompt"] of the target to return & newValue & return
    exit myFrenchStrings
    break
  case "filePrompt"
    set the myFrenchStrings["filePrompt"] of the target to return& newValue & return
    exit myFrenchStrings
    break
  default
    pass myFrenchStrings
  end switch
end myFrenchStrings
```

Comme vous pouvez le voir dans les structures de contrôle **exit**, **pass** et **end**, le nom de ce gestionnaire **setProp** est le même que le nom que le jeu de propriétés personnalisées qu'il contrôle - ***myFrenchStrings***. Parce qu'il n'y a qu'un seul gestionnaire pour toutes les propriétés personnalisées dans cet ensemble, le gestionnaire utilise la structure de contrôle **switch** pour effectuer une action différente pour chaque propriété qu'il traite.
Supposons que vous modifiez la propriété personnalisée ***standardErrorPrompt*** :

```
set the customPropertySet of this stack to "myFrenchStrings"
set the standardErrorPrompt of this stack to field 1
```

LiveCode envoie un déclencheur **setProp** à la pile, ce qui provoque l'exécution du gestionnaire ci-dessus. La propriété que vous définissez, - ***standardErrorPrompt*** - est placée dans le paramètre ***myPropertyName***, et la nouvelle valeur - le contenu du champ 1 - est placée dans le paramètre ***newValue***. Le gestionnaire exécute la condition de ***standardErrorPrompt***, mettant un caractère ***return*** avant et après la propriété, avant de la définir. Si vous définissez une propriété personnalisée autre que ***standardErrorPrompt*** ou ***filePrompt*** dans le jeu ***myFrenchStrings***, le cas par défaut est exécuté. En l'occurrence la structure de contrôle de la carte permet au déclencheur **setProp** continuer sur le chemin du message, et quand il atteint le moteur, LiveCode définit la propriété personnalisée.

Note : Comme mentionné ci-dessus, vous pouvez adresser une propriété personnalisée dans un ensemble soit par la première bascule sur ce jeu, soit en utilisant l'ecriture dans un tableau pour indiquer à la fois l'ensemble et des propriétés. Un exemple ci-dessous :

```
set the customPropertySet of me to "mySet"
set the myProperty of me to true
-- est équivalent à :
set the mySet["myProperty"] of me to true
```

Peu importe comment vous définissez la propriété personnalisée, si elle est membre d'un ensemble de propriétés personnalisées, le déclencheur **setProp** a le nom de cet ensemble - pas celui de la propriété personnalisée - et vous devez utiliser un gestionnaire **setProp** sous la forme décrite ci-dessus pour piéger le déclencheur **setProp**.

7.10.3 Répondre à une demande sur la valeur d'une propriété personnalisée

Lorsque vous utilisez une propriété personnalisée dans une expression, LiveCode envoie un appel **getProp** à l'objet dont la valeur de propriété est demandée.
Un appel **getProp** agit un peu comme un appel de fonction personnalisée. Il est envoyé à un objet particulier. Si le script de cet objet contient un gestionnaire **getProp** pour la propriété, le gestionnaire est exécuté, et LiveCode, substitue la valeur retournée à la référence de la propriété personnalisée. Sinon, l'appel se déplace le long du chemin de message jusqu'à ce qu'il trouve un gestionnaire pour cette propriété. Si l'appel de **getProp** atteint la fin du chemin de message sans être piégé, LiveCode, remplace la valeur de la propriété personnalisée, de l'expression.
Vous pouvez inclure autant de gestionnaires **getProp** dans un script que nécessaire.

La structure d'un gestionnaire getProp

Contrairement à un gestionnaire de messages, un gestionnaire **getProp** commence par le mot **getProp** à la place du mot **on**.
Il est suivi par le nom du gestionnaire (qui est le même que le nom de la propriété personnalisée). Un gestionnaire de **getProp**, comme tous les gestionnaires, se termine par le mot «**end**» suivi par le nom du gestionnaire.
L'exemple suivant est un gestionnaire **getProp** pour une propriété personnalisée appelée ***percentUsed*** :

```
getProp percentUsed
  global lastAccessTime
  put the seconds into lastAccessTime
  pass percentUsed
end percentUsed
```

Lorsque vous utilisez la propriété personnalisée **percentUsed** dans une expression, le gestionnaire est exécuté :

```
put the percentUsed of card 1 into myVariable
```

Lorsque cette instruction est exécutée, LiveCode envoie un appel par **getProp** à la carte pour récupérer la valeur de la propriété **percentUsed**. Cela exécute le gestionnaire par **getProp** pour la propriété. L'exemple du gestionnaire enregistre la date et l'heure actuelle dans une variable globale avant que la propriété ne soit évaluée. Pour plus de détails, voir **getProp** dans le dictionnaire de LiveCode.

Renvoi d'une valeur dans un gestionnaire de getProp

Lorsque le déclencheur **getProp** atteint le moteur - la dernière étape dans le chemin de message - LiveCode obtient la propriété personnalisée à partir de l'objet et substitue sa valeur dans l'expression, la où la propriété a été utilisée.

Pour laisser un déclencheur passer plus loin sur le chemin de message, utilisez la structure de contrôle **pass**. La structure de contrôle **pass**, arrête le gestionnaire en cours et envoie le déclencheur à l'objet suivant dans le chemin de message, comme si l'objet n'avait pas de gestionnaire pour la propriété personnalisée. Pour signaler une valeur autre que le valeur stockée dans la propriété personnalisée - par exemple, si vous souhaitez reformater la valeur, d'abord - vous utilisez la structure de contrôle **return** au lieu de passer l'appel **getProp**. L'exemple suivant est un gestionnaire **getProp** pour une propriété personnalisée appelée ***LastChanged*** qui détient une date en secondes :

```
getProp lastChanged
  get the lastChanged of the target
  convert it to long date
  return it
end lastChanged
```

La structure de contrôle **return**, lorsqu'elle est utilisée dans un gestionnaire **getProp**, rapporte une valeur de propriété au gestionnaire qui l'a demandé. Dans l'exemple ci-dessus, la date convertie est ce qui est rapporté, et non la propriété brute. Comme vous pouvez le voir dans l'exemple, vous n'êtes pas limité à retourner la valeur réelle stockée de la propriété personnalisée. En fait, vous pouvez retourner n'importe quelle valeur à partir d'un gestionnaire **getProp**.

Important : Si vous utilisez la valeur d'une propriété personnalisée, dans le gestionnaire **getProp** de la propriété, aucun appel de **getProp** n'est envoyé à l'objet cible. Il s'agit d'éviter un emballement récursif lorsque le gestionnaire **getProp** s'appelle.

Un gestionnaire peut soit utiliser la structure de contrôle **return** pour renvoyer une valeur, soit utiliser la structure de contrôle **pass** pour laisser LiveCode obtenir la propriété personnalisée à partir de l'objet. Si l'appel de **getProp** est piégé avant qu'il n'atteigne le moteur et qu'aucune valeur n'est retournée dans le gestionnaire **getProp**, la propriété personnalisée signale une valeur vide. En d'autres termes, un gestionnaire **getProp** doit inclure soit une structure de contrôle **return** soit une structure de contrôle **pass** ou alors sa propriété personnalisée sera toujours signalée comme vide.

Utilisation du chemin du message par un appel getProp

Parce que les appels **getProp** utilisent le chemin du message un seul objet peut recevoir les appels **getProp** pour tous les objets qui lui appartiennent. Par exemple, **getProp** appelle à ce que toutes les commandes sur une carte soient envoyées à la carte, si le script de la commande n'a pas de gestionnaire de cette propriété. Vous pouvez profiter du chemin de message à mettre en œuvre le même comportement **getProp** pour les objets qui ont tous la même propriété personnalisée.

Attention : Si un gestionnaire **getProp** n'est pas attaché à l'objet qui a la propriété personnalisée et utilise la valeur de la propriété personnalisée un emballement récursif se produira. Pour éviter ce problème, définissez la propriété **lockMessages** sur vrai avant d'obtenir la valeur de la propriété personnalisée.

Propriétés inexistantes

Si la propriété personnalisée spécifiée par un gestionnaire **getProp** n'existe pas, le gestionnaire **getProp** est toujours exécuté si la propriété est utilisée dans une expression. Les propriétés inexistantes retournent une valeur vide ; obtenir la valeur d'une propriété personnalisée qui n'existe pas ne provoque pas d'erreur de script.

Les jeux de propriétés personnalisées et les gestionnaires getProp

Un gestionnaire de **getProp** pour un ensemble de propriétés personnalisées se comporte différemment d'un gestionnaire **getProp** pour une propriété personnalisée qui est dans le set actif par défaut.
Lorsque vous utilisez la valeur d'une propriété personnalisée dans un jeu de propriétés personnalisées, l'appel **getProp** est nommé pour le jeu entier, non pour cette propriété. Le nom de la propriété est passée à un paramètre en utilisant la notation de type ***array***. Cela signifie que pour les propriétés personnalisées dans un ensemble vous écrivez un gestionnaire **getProp** unique pour cet ensemble plutôt que pour chaque propriété. L'exemple suivant gère les appels **getProp** pour toutes les propriétés personnalisées dans un jeu de propriété personnalisée appelée ***expertSettings*** qui contient des propriétés personnalisées nommées ***fileMenuContents***, ***editMenuContents***, et peut-être d'autres propriétés personnalisées :

```
getProp expertSettings[thePropertyName]    -- Le paramètre thePropertyName contient le nom de la propriété à modifier
  switch thePropertyName
  case "fileMenuContents"
    if the expertSettings[fileMenuContents] of the target is empty then
      return "No items"
    else pass expertSettings
    break
  case "editMenuContents"
    if the expertSettings[editMenuContents] of the target is empty then
      return the noviceSettings[editMenuContents] of the target
    else pass expertSettings
    break
  default
   pass expertSettings
  end switch
end expertSettings
```

Comme vous pouvez le voir sur les structures de contrôle **pass** et **end**, le nom du gestionnaire **getProp** est le même que nom du jeu de propriétés personnalisées qu'il contrôle - ***expertSettings***. Parce qu'il n'y a qu'un seul gestionnaire pour toutes les propriétés personnalisées dans cet ensemble, le gestionnaire utilise la structure de commande de commutateur (**switch**) pour effectuer une action différente pour chaque propriété qu'il traite.

Supposons que vous obtenez la propriété personnalisée, "***fileMenuContents***" :

```
set the customPropertySet of button 1 to "expertSettings"

put the fileMenuContents of button 1 into me
```

LiveCode envoie un appel de **getProp** au bouton ce qui provoque l'exécution du gestionnaire ci-dessus. La propriété que vous interrogez - ***fileMenuContents*** - est placée dans le paramètre ***thePropertyName*** Le gestionnaire exécute le cas pour ***fileMenuContents*** : si la propriété est vide, il renvoie **No items**. Sinon, la structure de contrôle **pass**, permet l'appel **getProp** de continuer sur le chemin du message, et quand il atteint le moteur, LiveCode obtient la propriété personnalisée.

7.11 Propriétés virtuelles

Une propriété virtuelle est une propriété personnalisée qui n'existe que dans un **setProp** et/ou dans le gestionnaire **getProp**, et n'est jamais réellement définie. Les propriétés virtuelles ne sont jamais attachées à l'objet. Au lieu de cela, elles agissent pour déclencher les gestionnaires **setProp** ou bien **getProp**, qui font le travail proprement dit.

Lorsque vous utilisez la commande **set** avec une propriété virtuelle, son gestionnaire **setProp** est exécuté, mais le déclenchement **setProp** n'est pas passé à la machine, de sorte que la propriété n'est pas attachée à l'objet. Lorsque vous utilisez une propriété virtuelle dans une expression, son gestionnaire **getProp** renvoie une valeur sans faire référence à l'objet. Dans les deux cas, l'utilisation de la propriété exécute simplement un gestionnaire.

Vous pouvez utiliser les propriétés virtuelles pour :

- Donner à un objet un ensemble de comportements
- Calculer la valeur d'un objet
- Mettre en œuvre une nouvelle propriété qui agit comme une propriété intégrée

7.11.1 Quand utiliser les propriétés virtuelles

Parce qu'elles ne sont pas stockées avec l'objet, les propriétés virtuelles sont transitoires : ce qui signifie qu'elles sont recalculées chaque fois que vous les demandez. Quand une propriété personnalisée dépend d'autres propriétés qui peuvent être définies de manière indépendante, il est approprié d'utiliser une propriété virtuelle.
Par exemple, le gestionnaire suivant calcule la position actuelle d'une barre de défilement en pourcentage (au lieu d'un nombre absolu) :

```
getProp asPercentage    -- de la barre de défilement
  put the endValue of the target - the startValue of the target into valueExtent
  return the thumbPosition of me * 100 div valueExtent
end asPercentage
```

La propriété personnalisée ***asPercentage*** dépend du ***thumbPosition*** de la barre de défilement, qui peut être modifié à tout moment (que ce soit par l'utilisateur ou par un gestionnaire). Pour cette raison, si nous fixons une propriété personnalisée pour l'objet, elle devra être recalculée chaque fois que la barre de défilement est actualisée afin de rester à jour. En utilisant une propriété virtuelle, vous pouvez vous assurer que la valeur de la propriété n'est jamais périmée, parce que le gestionnaire **getProp** recalcule chaque fois que vous appelez ***asPercentage*** de la barre de défilement.
Les propriétés virtuelles sont également des solutions utiles lorsque la valeur d'une propriété est grande. Parce que la propriété virtuelle n'est pas stockée avec l'objet, elle ne prend pas de place sur le disque, et n'occupe la mémoire que quand elle est calculée.

Une autre raison d'utiliser une propriété virtuelle est d'éviter la redondance. Le gestionnaire suivant définit la largeur d'un objet, non en pixels, mais en tant que pourcentage de la largeur (**width**) du propriétaire de l'objet :

```
setProp percentWidth newPercentage
  set the width of the target to the width of the owner of the target * newPercentage div 100
end percentWidth
```

Supposons que ce gestionnaire se trouve dans le script d'un bouton de la carte, dans une pile de 320 pixels de large. Si vous réglez l'attribut du bouton "percentWidth" à 25, la largeur du bouton est réglé sur 80, soit 25% de la largeur de 320 pixels de la carte. Toutefois, il n'y a pas beaucoup de sens à stocker la valeur **percentWidth** d'un objet, parce qu'il est basé sur la largeur de l'objet et la largeur de son propriétaire.

Pensez à utiliser les propriétés virtuelles lorsque vous souhaitez définir un attribut d'un objet, mais qu'il n'y pas de sens de stocker l'attribut avec l'objet - il serait redondant, parce que les changements possibles sur l'objet signifie qu'il devrait être re-calculé de toute façon, ou parce que la propriété est trop grande pour être facilement stockée.

7.11.2 Gestionnaires pour une propriété virtuelle

Comme vous pouvez le voir en regardant l'exemple ci-dessus, un gestionnaire pour une propriété virtuelle est structuré comme un gestionnaire pour toute autre propriété personnalisée. La seule différence structurelle est que, puisque le gestionnaire a déjà tout ce qu'il faut faire, il n'est pas nécessaire d'attacher réellement la propriété personnalisée à l'objet ou obtenir sa valeur de l'objet. Lorsque vous définissez une propriété virtuelle ou utilisez sa valeur, le déclencheur de **setProp** ou un appel **getProp** n'atteint pas le moteur, mais est piégé par un gestionnaire d'abord.

Propriété Virtuelle : commande setProp

Une commande **setProp** pour une propriété personnalisée ordinaire comprend la structure de contrôle de passage, permettant le déclenchement de **setProp** pour atteindre le moteur et le réglage de la propriété personnalisée (ou bien il comprend une commande **set** qui définit directement la propriété). De plus, un gestionnaire d'une propriété virtuelle ne comprend pas la structure de contrôle de passage, car une propriété virtuelle ne doit pas être réglée.
Puisque la propriété est définie automatiquement lorsque le déclencheur atteint la fin du chemin du message, le gestionnaire d'une propriété virtuelle ne transmet pas le déclencheur. Si vous examinez les propriétés personnalisées d'un objet après avoir réglé une propriété virtuelle, vous verrez que la propriété personnalisée n'a pas été réellement créée. Cela arrive parce que le gestionnaire **setProp** piège l'appel pour définir la propriété (à moins d'activer le déclencheur de **setProp)** donc la propriété n'est pas passée à LiveCode, et n'est pas définie.

Propriété Virtuelle : commande getProp

De même, que pour un gestionnaire **getProp**, pour une propriété personnalisée ordinaire, soit obtient la valeur de la propriété directement soit transmet l'appel **getProp** pour que le moteur puisse retourner la valeur de la propriété. Mais dans le cas d'une propriété virtuelle, l'objet ne comporte pas la propriété, de sorte que le gestionnaire **getProp** doit renvoyer une valeur.

7.11.3 Création de nouvelles propriétés de l'objet

Vous pouvez utiliser les propriétés virtuelles pour créer une nouvelle propriété qui s'applique à tous les objets, ou à tous les objets d'un type particulier. Une telle propriété agit comme une propriété intégrée, parce que vous pouvez l'utiliser pour n'importe quel objet. Et parce qu'une propriété virtuelle ne repose pas sur une propriété personnalisée stockée dans l'objet, vous n'avez pas besoin de préparation en créant la propriété pour chaque nouvel objet que vous créez : la propriété virtuelle n'est calculée que si vous l'utilisez dans une expression. .
L'exemple suivant illustre comment implémenter une propriété virtuelle appelée "percentWidth" qui se comporte comme une propriété intégrée.

Réglage de la propriété "percentWidth"

Supposons que vous placiez le gestionnaire **percentWidth** décrit ci-dessus dans un script de pile plutôt que dans le script d'un bouton :

```
setProp percentWidth newPercentage
  set the width of the target to the width of the owner of the target * newPercentage div 100
end percentWidth
```

Parce le déclencheur **setProp** utilise le chemin du message, si vous définissez le **percentWidth** de tout objet dans la pile, la pile entière reçoit le déclenchement de **setProp** (sauf s'il est pris au piège par un autre objet en premier). Cela signifie que si le gestionnaire est dans le script de la pile, vous pouvez définir la propriété **percentWidth** de tout objet dans la pile.

Si vous placez ce gestionnaire dans un ***backscript***, vous pouvez définir la "**percentWidth**" de tout objet, n'importe où dans l'application.

Note : Pour faire référence à l'objet dont la propriété est définie, utilisez la fonction **target**. Elle réfère à l'objet qui a reçu le premier déclencheur de **setProp** - l'objet dont la propriété personnalisée est le réglage de - même si le gestionnaire est en cours d'exécution dans le script d'un autre objet.

Obtenir la propriété "percentWidth"

Le gestionnaire **getProp** correspondant, qui vous permet de récupérer le **percentWidth** d'un objet, ressemble à ceci :

```
getProp percentWidth
  return 100 * (the width of the target div the width of the owner of the target)
end percentWidth
```

Si vous placez le gestionnaire du dessus uniquement dans le script d'un bouton de la carte, la déclaration suivante retourne la largeur du bouton en pourcentage :

```
put the percentWidth of button "My Button" into field 12
```

Par exemple, si la pile est de 320 pixels de large et le bouton est de 50 pixels de large, la largeur du bouton est de 15% de la largeur de la carte, et l'instruction place "15" dans le champ.

Comme le gestionnaire **setProp** de cette propriété, le gestionnaire **getProp** devrait être placé le long du chemin du message. Le mettre dans un script de pile rend la propriété accessible à tous les objets dans la pile ; le mettre dans un **backscript** rend la propriété accessible à tous les objets de l'application.

Limitation de la propriété percentWidth

La plupart des propriétés intégrées ne s'appliquent pas à tous les types d'objets, et vous pouvez également créer une propriété virtuelle qui ne s'applique qu'à certains types d'objets. Par exemple, il n'est pas très utile pour obtenir la largeur d'une sous-pile en tant que pourcentage de sa pile principale, ou la largeur d'une carte en tant que pourcentage de la largeur de la pile.

Vous pouvez limiter la propriété de certains types d'objets en vérifiant le nom de l'objet cible :

```
setProp percentWidth newPercentage
  if word 1 of the name of the target is "stack" or word 1 of the name of the target is "card" then
    exit setProp
  else
    set the width of the target to the width of the owner of the target * newPercentage div 100
  end if
end percentWidth
```

Le premier mot du nom d'un objet est le type d'objet, de sorte que le gestionnaire révisé ci-dessus ignore le réglage **percentWidth** si l'objet est une carte ou une pile.

7.12 Gérer les fenêtres, les palettes et les boîtes de dialogue

LiveCode fournit un contrôle complet sur tous les aspects de la gestion des fenêtres, y compris le déplacement, la superposition, et le changement de mode fenêtré.

7.12.1 Déplacement d'une fenêtre

Habituellement, vous utilisez la propriété **location** ou **rectangle** d'une pile pour déplacer la fenêtre de la pile. La propriété **location** indique le centre de la fenêtre de la pile, par rapport au coin supérieur gauche de l'écran principal. Contrairement à **location** des contrôles, **location** d'une pile est spécifié en coordonnées absolues. La déclaration suivante déplace une pile au centre de l'écran principal :

```
set the location of stack "Wave" to the screenLoc
```

La propriété **rectangle** d'une pile indique la position des quatre bords, et peut être utilisée pour redimensionner la fenêtre ainsi que la déplacer :

```
set the rectangle of this stack to "100,100,600,200"
```

Astuce : Pour ouvrir une fenêtre à un endroit particulier, sans scintillement, réglez l'emplacement de la pile ou de la propriété **rectangle** à la valeur désirée soit avant d'y aller soit dans le gestionnaire **preOpenStack** de la pile.

Vous pouvez également utiliser les propriétés associées pour déplacer une fenêtre. Changer la propriété **bottom** ou **top** d'une pile déplace la fenêtre vers le haut ou vers le bas sur l'écran. Modifier la propriété **left** ou **right** déplace la fenêtre de droite à gauche.

7.12.2 Modifier le calque d'une fenêtre

Vous ramenez une fenêtre au premier plan à l'aide de la commande **go** :

```
go stack "Alpha"
```

Si la pile est déjà ouverte, la commande **go** la ramène à l'avant, sans changer son mode.

Pour connaître l'ordre des couches des fenêtres de pile ouvertes, utilisez la fonction **openStacks**. Cette fonction répertorie toutes les fenêtres de pile ouvertes dans l'ordre d'avant en arrière.

La couche palette

Normalement, les fenêtres palette flottent au-dessus des fenêtres éditables et boîtes de dialogue modales. Une palette sera toujours au-dessus d'une fenêtre standard, même si vous ramenez la fenêtre standard à l'avant. Cela permet de s'assurer que les palettes, qui contiennent généralement des outils à utiliser depuis n'importe quelle fenêtre, ne risquent pas disparaître derrière des fenêtres de document.
C'est aussi une bonne raison de vous assurer que les fenêtres palette restent petites, parce que les autres fenêtres ne peuvent être déplacées si la palette, bloque une partie de la fenêtre.

La couche palette système

Les fenêtres système - piles dont la propriété **systemWindow** est vraie - flottent au-dessus toutes les autres fenêtres, dans toutes les applications en cours d'exécution.
Cela signifie que même si l'utilisateur met une autre application à l'avant, les fenêtres système de votre application restent en avant de toutes les fenêtres.
Les fenêtres système, sont toujours devant les autres fenêtres, et vous ne pouvez pas changer ce comportement.

7.12.3 La fenêtre active

Dans la plupart des applications, les commandes sont appliquées sur la fenêtre active. Puisque LiveCode vous donne la possibilité d'utiliser plusieurs types de fenêtres, toutes n'étant pas modifiables, la pile actuelle

n'est pas toujours identique à celle de la fenêtre active. La pile courante est la cible du choix du menu comme **View > Go Next** et la pile est indiquée par l'expression **this stack**.
Par exemple, l'exécution de la commande **find** peut avoir des résultats inattendus si des piles de modes différents sont ouvertes parce que dans ces conditions, la recherche peut cibler une pile qui n'est pas la fenêtre active.

Trouver la pile actuelle

La pile actuelle - la pile qui répond aux commandes - est désignée par la propriété **defaultStack**.
Pour déterminer quelle pile est la pile actuelle, utilisez les règles suivantes :

1. Si des piles sont ouvertes dans une fenêtre modifiable, la pile actuelle est la pile déverrouillée de premier plan. (Une pile est déverrouillé si sa propriété **cantModify** est définie sur faux.)
2. S'il n'y a pas de piles non verrouillées ouvertes, la pile actuelle est la pile verrouillée premier plan dans une fenêtre modifiable.
3. S'il n'y a pas de piles ouvertes dans une fenêtre modifiable, la pile courante est la pile le plus en avant dans une boîte de dialogue non modale.
4. S'il n'y a pas fenêtres ouvertes, modifiables ou non modales, la pile actuelle est la palette de premier plan.

Une autre façon d'exprimer cet ensemble de règles est de dire que la pile actuelle est la pile de premier plan avec la propriété **mode** au niveau le plus bas. Vous pouvez savoir quelle pile a le mode le plus bas en utilisant la fonction **topStack**.

La fonction topStack et la propriété defaultStack :

La propriété **defaultStack** précise quelle est la pile actuelle. Par défaut, **defaultStack** est réglé sur **topStack**, mais vous pouvez paramétrer **defaultStack** sur toute pile ouverte.

La fonction **topStack** passe par les piles ouvertes, d'abord par **mode**, puis par couches. Par exemple, si toutes les fenêtres modifiables sont ouvertes, la fenêtre modifiable la plus élevée est **topStack**. S'il n'y a pas de fenêtres modifiables, **topStack** représente la boîte de dialogue non modale plus élevée, et ainsi de suite.

Changer la pile actuelle

Pour utiliser une pile différente de la pile actuelle, définir la propriété **defaultStack** sur la pile que vous souhaitez cibler avant d'exécuter les commandes. Habituellement, le **defaultStack** est le **topStack**, mais vous pouvez changer ceci si vous voulez remplacer les règles habituelles sur la fenêtre active.

Une note sur les systèmes Unix

Si votre système est configuré pour utiliser le focus pointeur plutôt que le click ou focus explicite, vous pouvez rencontrer des résultats inattendus lors de l'utilisation LiveCode, puisque la pile actuelle va changer lorsque vous déplacez le pointeur de la souris. Il est recommandé de configurer votre système pour utiliser le focus explicite lors de l'utilisation LiveCode ou d'autres applications créées dans LiveCode.

7.12.4 Création d'une toile de fond

Pour certaines applications, vous pouvez créer un fond uni ou à motifs en arrière plan des fenêtres de votre application. Cette toile de fond empêche les fenêtres d'autres applications d'être vues - même si elle ne ferme pas les fenêtres. Ceci est donc approprié pour des applications comme un jeu ou un kiosque, où l'utilisateur n'a pas besoin de voir d'autres applications et où vous voulez maintenir le niveau de distraction à un minimum.

Pour créer un fond, vous définissez la propriété **backdrop** soit à une référence de couleur valide, ou l'**ID** de l'image que vous voulez utiliser comme une mosaïque :

```
set the backdrop to "#99FF66"  -- une couleur
set the backdrop to 1943  -- un identifiant image
```

Dans l'environnement de développement de LiveCode, vous pouvez créer un fond en sélectionnant **View > Backdrop**. Utilisez la boîte de dialogue Préférences pour spécifier une couleur de fond à utiliser.

7.12.5 Fenêtres Ouvertes, Fermées et Masquées

Chaque pile ouverte est affichée dans une fenêtre de la pile. Une pile peut être ouverte sans être visible, et peut être chargée en mémoire sans être ouverte.

Piles cachées

Une fenêtre de la pile peut être soit affichée ou masquée, selon la propriété **visible** de la pile. Cela signifie qu'une fenêtre peut être ouverte sans être visible sur l'écran.

Astuce : Pour lister toutes les piles ouvertes, qu'elles soient visibles ou masquées, utilisez la fonction de **openStacks**.

Piles chargées

Une pile peut également être chargée en mémoire sans être réellement ouverte. Une pile dont la fenêtre est fermée (pas seulement cachée) n'est pas répertoriée par la fonction **openStacks.**

Cependant, cela occupe de la mémoire, et ses objets sont accessibles à d'autres piles.
Par exemple, si une pile fermée et chargée en mémoire contient une certaine image, vous pouvez utiliser l'image comme une icône de bouton dans une autre pile.

Une pile peut être chargée dans la mémoire sans être ouverte sous l'une des conditions suivantes :

Un gestionnaire d'une autre pile fait référence à une propriété de la pile fermée. Ceci charge automatiquement la pile référencée dans la mémoire.

La pile est dans le même fichier de pile que l'autre pile qui est ouverte.

La pile a été ouverte puis refermée, et sa propriété **destroyStack** est définie sur faux. (Si la propriété **destroyStack** est faux, la pile est fermée mais non déchargée, lorsque sa fenêtre est fermée.)

Astuce : Pour lister toutes les piles en mémoire, qu'elles soient ouvertes ou fermées, utiliser la fonction **revLoadedStacks**.

7.12.6 Les états d'une pile

Une pile, donc, peut-être dans l'un des quatre états :

- **Ouverte et visible :** la pile est chargée en mémoire, sa fenêtre est ouverte, et la fenêtre est visible.
- **Ouverte et cachée :** La pile est chargée en mémoire, sa fenêtre est ouverte, mais la fenêtre est masquée. La pile est répertorié dans le menu Fenêtre et dans l'Explorateur d'applications.
- **Fermée mais chargée en mémoire :** La pile est chargée en mémoire, mais sa fenêtre n'est pas ouverte et n'est pas répertoriée par la fonction **openStacks** ou dans le menu Fenêtre. Cependant, ses objets sont toujours disponibles pour les autres piles, et elle est listée dans le navigateur de l'application. Une pile qui est fermée mais chargée en mémoire possède une propriété **mode** à zéro.

Pour supprimer ce type de pile de la mémoire, sélectionnez **Tools** > **Application Browser**, trouver le nom de la pile, et **contrôle-clic** (Mac OS ou OS X) ou un **clic droit** sur celui-ci (Unix ou Windows) dans la fenêtre du navigateur d'application et choisissez **Close and Remov from Memory** dans le menu contextuel.

- **Fermée :** La pile n'est pas chargée en mémoire et n'a aucun effet sur les autres piles.

7.12.7 Types de fenêtres et la propriété mode

Via un script, vous pouvez savoir le type d'une fenêtre de la pile en vérifiant la propriété **mode** de la pile. Cette propriété en lecture seule indique un nombre qui dépend du type de fenêtre. Par exemple, la propriété **mode** d'une fenêtre modifiable est 1, et la propriété **mode** d'une palette est 4.
Vous pouvez utiliser la propriété **mode** dans un script pour vérifier quel type de fenêtre d'une pile s'affiche :

```
if the mode of this stack is 5 then -- modal dialog box
   close this stack
else -- some other type of window
   beep
end if
```

Pour des informations complètes sur les valeurs possibles de la propriété **mode**, voir son entrée dans le Dictionnaire de LiveCode.

7.12.8 Apparence de la fenêtre

Les détails de l'apparence d'une fenêtre, comme la hauteur de sa barre de titre et la couleur de fond ou le motif de la fenêtre elle-même, sont essentiellement déterminées par le mode de la pile. Il ya quelques éléments supplémentaires de l'apparence de la fenêtre que vous pouvez contrôler avec des propriétés spécifiques.
La propriété **metal :** sur les systèmes OS X, vous utilisez la propriété **metal** d'une pile pour donner à la fenêtre de la pile un aspect métallique texturé. Cet aspect métallique s'applique à la barre de titre de la pile et à son arrière-plan.

Astuce : L'aspect métallique, en général, doit être utilisé uniquement pour la fenêtre principale de l'application, et uniquement pour les fenêtres qui représentent un dispositif de média physique tel qu'un lecteur CD. Voir les lignes directrices de l'interface utilisateur d'Apple pour plus d'informations.

Couleur de fond de la fenêtre

La couleur de fond ou le motif de la zone de contenu de la fenêtre - la partie qui n'est pas une partie de la barre de titre - est déterminé par le type de fenêtre et le système d'exploitation, par défaut.
Si vous voulez une fenêtre ayant une couleur ou un motif spécifique, vous pouvez définir la propriété **backgroundColor** ou **backgroundPattern** de la pile :

```
set the backgroundColor of stack "Alpha" to "aliceblue"
set the backgroundPattern of stack "Beta" to 2452 -- img ID
```

Cette une couleur ou ce motif écrase la couleur habituelle ou le motif de fond.

7.12.9 La propriété *Decorations*

La plupart des propriétés qui se rapportent à l'apparence d'une fenêtre peuvent également être définies dans la propriété propriété **decorations** de la pile. La propriété **decorations** d'une pile est constituée d'une liste des ornements, séparés par des virgules :

```
set the decorations of stack "Gamma" to "title,minimize"
```

La déclaration ci-dessus définit propriété **minimizeBox** de la de la pile comme vraie, ainsi que l'affichage de sa barre de titre, et fixe d'autres propriétés de la pile (**MaximizeBox**, **CloseBox**, **metal**) comme fausses.
Inversement, si vous définissez la propriété **minimizeBox** de la pile comme vraie, sa propriété **decorations** est modifiée pour inclure **minimize** comme l'un de ses attributs. De cette façon, la propriété **decorations** d'une pile interagit avec son **CloseBox**, **MinimizeBox**, **ZoomBox**, **metal**, **Shadow** et les propriétés de **systemWindow**.

La propriété *decorations* et les barres de menus dans une fenêtre

Sur les systèmes Unix et Windows, la barre de menus apparaît dans la partie supérieure de la fenêtre. Sur ces systèmes, lorsque une fenêtre affiche sa barre de menu ceci est déterminé par le fait que la propriété des **decorations** de la pile comprend "menu" :

```
set the decorations of this stack to "title,menu"
```

Sur Mac OS X, la barre de menus apparaît en haut de l'écran, en dehors de toute fenêtre. Sur ces systèmes, la décoration "menu" n'a aucun effet.

Barre Titre

L'utilisateur fait glisser la barre titre d'une fenêtre pour déplacer la fenêtre sur l'écran. En général, si la barre titre n'est pas affichée, l'utilisateur ne peut pas déplacer la fenêtre. Vous utilisez la propriété **decorations** (voir ci-dessous) pour afficher et masquer la barre titre d'une fenêtre.
Lorsque l'utilisateur fait glisser la fenêtre, LiveCode envoie un message de **moveStack** à la carte utilisée.
La propriété **decorations** ne concerne que la fenêtre qui peut être déplacée en la faisant glisser. Même si la propriété **decorations** d'une pile ne comprend pas la **decoration** barre titre, vous pouvez toujours définir les **location** et **rectangle** d'une pile, et les propriétés associées pour déplacer ou redimensionner la fenêtre.

Titre de la fenêtre

Le titre qui apparaît dans la barre de titre d'une fenêtre est déterminée par la propriété **label** (étiquette) de la pile. Si vous modifiez **label** d'une pile dans un script, le titre de la fenêtre est immédiatement mis à jour.

Si le **label** est vide, la barre de titre affiche la propriété **name** de la pile. (Si la pile est dans une fenêtre éditable dont **cantModify** est sur faux, un astérisque apparaît après le titre de la fenêtre pour indiquer cela, et si la pile a plus d'une carte, le numéro de carte apparaît également dans le titre de la fenêtre. Ces indicateurs n'apparaissent pas si la pile a une propriété **label**).

Parce que le titre de la fenêtre est déterminée par la propriété de la pile **label** au lieu de sa propriété **name**, vous avez beaucoup de souplesse pour modifier titre de la fenêtre. Vos scripts font référence à la pile par son **name** - qui n'a pas besoin de changer - pas par son **label**, ainsi vous pouvez changer le titre de la fenêtre sans modifier les scripts qui font référence à la pile.

Le bouton de fermeture

Le bouton de fermeture permet à l'utilisateur de fermer la fenêtre en cliquant dessus. Pour masquer ou afficher le bouton de fermeture, vous définissez la propriété **CloseBox** de la pile :

```
set the closeBox of stack "Bravo" to false
```

Lorsque l'utilisateur clique sur le bouton de fermeture, LiveCode envoie un message de **closeStackRequest**, suivie d'un message de **closeStack**, à la carte en cours.

La propriété **CloseBox** ne concerne que la fenêtre qui peut être fermée en cliquant dessus. Même si la propriété **CloseBox** d'une pile est sur faux, vous pouvez toujours utiliser la commande **close** dans un gestionnaire ou la boîte de message pour fermer la fenêtre.

La boîte minimiser ou boîte de rétreint

La terminologie et le comportement de cette partie de la barre de titre varie selon la plateforme. Sur les systèmes Mac OS, la boîte ***collapse*** rétrécit la fenêtre de sorte que sa barre de titre est affichée. La boîte ***minimize*** (OS X et systèmes Windows) ou la boîte ***iconify*** (systèmes Unix) rétrécit la fenêtre à une icône sur le bureau.

Pour masquer ou afficher la boîte ***minimize*** ou la boîte ***collapse***, vous définissez la propriété **minimizeBox** de la pile:

```
set the minimizeBox of this stack to true
```

Astuce : Sur OS X et les systèmes Unix, vous pouvez définir la propriété de l'icône d'une pile pour indiquer l'icône qui apparaît lorsque la pile est réduite au minimum.

Lorsque l'utilisateur clique sur la case boîte de réduction ou minimiser, LiveCode envoie un message d'**iconifyStack** à la carte actuelle.

La boîte de maximiser ou la boîte zoom

La terminologie et le comportement de cette partie de la barre de titre varie selon la plateforme. Sous Mac OS, la boîte zoom bascule entre la fenêtre à sa taille actuelle et la taille maximale.
La boîte maximiser (Unix et Windows) élargit la fenêtre à sa taille maximale.
Pour masquer ou afficher la boîte zoom ou la boîte maximiser, vous définissez la propriété **zoomBox** de la pile :

```
set the zoomBox of stack "Hello" to false
```

Lorsque l'utilisateur clique sur la zone de zoom ou la zone maximiser, LiveCode envoie un message de **resizeStack** à la carte actuelle

7.12.10 Réaliser une pile redimensionnable

La propriété **resizable** d'une pile détermine si l'utilisateur peut modifier sa taille en faisant glisser un coin ou un bord (selon le système d'exploitation) de la fenêtre de la pile.

Astuce : Pour déplacer et redimensionner les contrôles automatiquement ajustés quand une pile est redimensionnée, utilisez la fenêtre **Geometry** dans l'inspecteur du contrôle de la propriété.

Certains modes de pile ne peuvent pas être redimensionnés, indépendamment de la valeur de la propriété **resizable** de la pile. Les boîtes de dialogue modales, les feuilles et les tiroirs ne peuvent pas être redimensionnés par l'utilisateur, et ne présentent pas de zone de redimensionnement.

La propriété **resizable** affecte uniquement la fenêtre qui peut être redimensionnée en faisant glisser un coin ou un bord.

Même si la propriété **resizable** d'une pile est définie sur faux, vous pouvez toujours définir les **location** et

rectangle, d'une pile et les propriétés associées pour déplacer ou redimensionner la fenêtre.

Lorsque l'utilisateur redimensionne une pile, LiveCode envoie un message de **resizeStack** à la carte en cours.

7.12.11 Fenêtres de formes irrégulières et translucides

Vous pouvez définir la propriété **windowShape** d'une pile sur le canal transparent ou alpha d'une image qui a été importée avec son canal alpha. Cela vous permet de créer une fenêtre avec des «trous» ou une fenêtre avec une transparence variable. Vous pouvez appliquer une forme à tout type de pile, quel que soit le mode dans lequel elle est ouverte, permettant à telle fenêtre un comportement modal de type dialogue, ou flottant comme une palette, etc

Vous pouvez utiliser une image GIF ou PNG pour les fenêtres de forme irrégulière. Si vous voulez la transparence vous devez utiliser des images PNG. La translucidité est actuellement pris en charge uniquement sur les systèmes Mac OS X et Windows.

7.13 Menus de Programmation et les Barres de Menus

Les menus dans LiveCode ne sont pas un type d'objet distinct. Au lieu de cela, vous créez un menu à partir d'un bouton ou d'une pile, puis utilisez les commandes spéciales pour afficher le menu ou l'inclure dans une barre de menu.

Cette rubrique traite :

- des barres de menus,
- des menus qui ne sont pas dans la barre de menu (comme les menus contextuels, les menus pop-up, et les menus d'options),
- comment faire des menus avec des caractéristiques spéciales telles que des coches et sous-menus,
- comment utiliser une fenêtre de la pile comme un Menu pour un contrôle total sur l'apparence du menu.

Pour créer facilement des barres de menu qui fonctionnent pour une application multiplate-forme, choisissez **Tools > Menu Builder**.
Voir la section sur le **Menu Builder** dans le chapitre sur la construction d'une interface utilisateur pour plus de détails. Les détails sur les barres de menu de cette rubrique ne sont nécessaires que si vous voulez modifier les barres de menu par script, par exemple si vous voulez inclure des fonctionnalités spécifiques non pris en charge par le **Menu Builder**.

7.13.1 Types de Menu

LiveCode prend en charge plusieurs types de menus : menus déroulants ***pulldown***, menus d'options ***option*** (habituellement appelés menus surgissants sous Mac OS et OS X), menus ***popup*** (habituellement appelés menus contextuels sous Mac OS et OS X), et combinés ***combo boxes***.

Chacun de ces types de menu est mis en œuvre par la création d'un bouton. Si la propriété du style du bouton est réglé sur **menu**, cliquer dessus fait apparaître un menu. La propriété **menuMode** du bouton détermine le type de menu qui est affiché.

Même les barres de menus sont créées en faisant un bouton menu déroulant pour chaque menu, puis un regroupement de ces boutons pour créer une barre de menu.
La barre de menu peut être déplacée vers le haut de la fenêtre de la pile (sur les systèmes Unix et Windows).

Pour afficher la barre de menu à l'emplacement standard au haut de l'écran sur Mac OS et les systèmes OS X, vous définissez la propriété **menubar** de la pile pour le nom du groupe. Le nom de chaque bouton est affiché dans la barre de menus sous forme de menu, et en déroulant vers le bas un menu affiche le contenu

du bouton comme une liste d'éléments de menu.

7.13.2 Boutons Menus

Vous pouvez créer un bouton menu en faisant glisser l'une des commandes de menu à partir de la palette d'outils. Toutefois, si vous souhaitez en créer un par script, le plus simple est de créer un bouton et de définir le style du bouton comme **menu**.

Ensuite, vous pouvez régler la propriété **menuMode** du bouton pour le type de menu approprié. Vous pouvez soit définir le **menuMode** dans un gestionnaire, ou utiliser l'option **Type menu** dans l'inspecteur des propriétés du bouton pour définir le type de menu.

Pour créer les items de menu qui s'affichent dans le menu, définissez la propriété de texte du bouton pour le contenu du menu, avec une option de menu par ligne.

Vous pouvez définir cette propriété dans un gestionnaire, ou remplir la case **menu elements** de la fenêtre des propriétés de base de l'inspecteur des propriétés.

Lorsque vous cliquez sur le bouton, le type de menu spécifié apparaît, avec le texte que vous avez entré qui s'affiche en tant qu'items dans le menu.

Astuce : Pour modifier dynamiquement le contenu du menu au moment où il est affiché, mettez un gestionnaire **mouseDown** dans le script du bouton qui place les items de menu dans le bouton. Lorsque le menu apparaît, il affiche les nouveaux éléments de menu.

Pour les menus qui conservent un état (comme les menus d'options et combinés), la propriété **label** du bouton garde le texte de l'élément de menu actuellement sélectionné.

Traiter le message menuPick

Lorsque l'utilisateur choisit un élément dans le menu, LiveCode envoie le message **menuPick** au bouton. Le paramètre de message est le nom de l'élément de menu choisi.
Si vous souhaitez effectuer une action lorsque l'utilisateur choisit un élément de menu, placez un gestionnaire du **menuPick** comme celui-ci dans le script du bouton :

```
on menuPick theMenuItem
  switch theMenuItem
    case "Name of First Item"
     -- do stuff here for first item
      break
    case "Name of Second Item"
     -- do stuff here for second item
      break
    case "Name of Third Item"
     -- do stuff here for third item
      break
  end switch
end menuPick
```

Modification de l'option de menu actuellement sélectionné

Pour les menus qui conservent un état (comme les menus d'options et combos), vous pouvez changer l'option de menu actuellement sélectionné en modifiant la propriété **label** de la touche pour le texte de la nouvelle option choisie.

Si vous changez l'option de menu actuellement sélectionné dans les menus d'options, définissez également la propriété **menuHistory** de la touche pour le numéro de ligne de la nouvelle option choisie. Cela garantit que le nouveau choix sera celui sous le pointeur de la souris la prochaine fois que l'utilisateur clique sur le menu.

7.13.3 Création de menus en cascade

Pour créer un menu en cascade (aussi appelé un sous-menu, menu déroulant à droite, ou un menu hiérarchique), ajouter un caractère de tabulation au début des éléments de menu que vous souhaitez placer dans le sous-menu.
Par exemple, le texte suivant, lorsqu'il est placé dans un bouton de menu, crée deux éléments de menu, puis un sous-menu contenant deux éléments, et enfin un dernier élément de menu :

Premier Item

Second Item

Troisième Item est un Submenu

    Premier Item dans Submenu

    Second Item dans Submenu

dernier Item du Menu n'est pas dans Submenu

La profondeur d'un élément de sous-menu est déterminé par le nombre de caractères de tabulation avant le nom de l'élément de menu. Le sous-menu devient une partie de la ligne la plus proche au-dessus de l'élément de sous-menu qui a un moins important nombre de caractère de tabulation.
Cela signifie que la première ligne d'un menu ne peut pas commencer par un caractère de tabulation, et chaque ligne dans le texte du bouton peut avoir au plus un caractère de tabulation plus que la ligne précédente.

Important : Vous ne pouvez pas créer une zone de liste déroulante en cascade et des menus d'options en cascade ne fonctionnent pas correctement sur toutes les plateformes. En général, vous devez créer des menus en cascade que dans le cadre d'un menu déroulant.

Cascade de menus et le message menuPick

Lorsque l'utilisateur choisit un élément de menu dans un menu en cascade, le paramètre du message **menuPick** contient le nom de l'élément de menu et le nom du sous-menu dont il fait partie, séparés par une **|** (barre verticale). Par exemple, si l'utilisateur choisit le deuxième élément du sous-menu dans le menu décrit ci-dessus, le paramètre envoyé avec le message **menuPick** est :

```
Troisième Item est un Submenu|Second Item dans Submenu
```

7.13.4 Puces, Tirets et Repères dans les menus

Il y a plusieurs caractères spéciaux que vous pouvez mettre au début d'une ligne, dans le contenu du bouton, pour modifier le comportement de l'élément de menu :

- **-** un tiret sur une ligne crée par lui-même une ligne de séparation
- **!c** crée une coche sur l'item de menu
- **!n** décoche le menu item
- **!r** place un diamant en entête de l'item de menu
- **!u** supprime le diamant

Si vous incluez les caractères spéciaux indiqués ci-dessus dans un sous-menu, le caractère spécial doit être placé au début de la ligne - devant les caractères de tabulation qui en font un élément de sous-menu.

Note : Vous ne pouvez pas créer des lignes de séparation dans les **combo boxes** ou dans les menus d'options sur les systèmes Windows.

Il y a trois autres caractères spéciaux qui peuvent apparaître n'importe où sur une ligne :

Mettre le caractère **&** n'importe où dans une ligne souligne le caractère suivant et rend le mnémonique du clavier pour cet élément de menu sur les systèmes Windows. Le caractère **&** n'apparaît pas dans le menu, et n'est pas envoyé avec le paramètre du message **menuPick** lorsque vous choisissez l'option dans un menu.

Mettre le caractère **/** n'importe où dans une ligne fait du caractère suivant le raccourci clavier pour l'élément de menu. Ni l' **/** ni le caractère suivant apparaissent dans le menu, pas plus qu'ils ne s'affichent dans le paramètre au message **menuPick**.

Pour mettre un caractère **&** ou **/** dans le texte d'un menu, doubler les caractères : **&&** ou **/ /**.
Mettre le caractère **(** n'importe où dans une ligne désactive le menu.
Pour mettre un caractère **(** dans un élément de menu sans le désactiver, le précéder d'une barre oblique inverse : **\ (**.

Remarque : Vous ne pouvez pas désactiver des lignes dans les comboboxes ou dans les menus d'options sur les systèmes Windows.

Tous les caractères spéciaux ci-dessus sont filtrés du paramètre envoyé avec le message **menuPick** lorsque l'utilisateur choisit un élément de menu. Le paramètre est le même que les caractères qui sont réellement affichés dans le menu.

Note : La police et la couleur d'un bouton de menu sont déterminés par les propriétés de police et de couleur du bouton. Cependant, sur les systèmes Mac, la police et la couleur des menus d'options et menus contextuels sont contrôlés par les paramètres du système d'exploitation, plutôt que par les propriétés de police et de couleur du bouton, si le **LookAndFeel** est réglé sur "**Appearance Manager**».

Activation et désactivation des éléments de menu

Pour activer ou désactiver un élément de menu dans un gestionnaire, vous pouvez ajouter ou supprimer le caractère spécial **(** mais il est généralement plus facile d'utiliser les commandes **enable menu** et **disable menu**:

```
enable menuItem 3 of button "My Menu"
disable menuItem 4 of me
```

Ces commandes ajoutent simplement ou suppriment le caractère spécial (au début de la ligne désignée par le contenu du bouton.

7.13.5 Barres de Menu sur Unix et Windows

Une barre de menu est composée d'un groupe de boutons de menu, avec la propriété **menuMode** de chaque bouton réglée sur "**pulldown**".

Astuce : Menu Builder permet de créer automatiquement une barre de menus pour vous. Pour utiliser le Générateur de menu, sélectionnez **Tools > Menu Builder**.

Pour créer une barre de menu à la main sans l'aide du Générateur de Menu :

1. Créer un bouton pour chaque menu, définir le style de chaque bouton menu, et réglez le **menuMode** du bouton, sur **pulldown** (déroulant). Vous pouvez définir ces propriétés dans un gestionnaire, ou tout simplement choisir **Object** > **New Control / Pulldown Menu** pour créer chaque bouton.
2. Mettez les éléments de menu dans le contenu de chaque bouton. Dans le script de chaque bouton, créer un gestionnaire **menuPick** pour effectuer toutes les actions que vous voulez faire quand un élément du menu est choisi.
3. Sélectionnez les boutons et les rassembler dans un groupe, puis déplacez le groupe à la position appropriée dans la partie supérieure de la fenêtre. Pour les systèmes Windows, réglez la propriété **textFont** du groupe à la police standard pour les menus de Windows, **MS Sans Serif**.

Important : Les boutons de votre barre de menu ne doivent pas se chevaucher. Un chevauchement des boutons peut entraîner un comportement inattendu lorsque l'utilisateur tente d'utiliser un menu.

7.13.6 Barres de Menus sur les Systèmes Mac OS

Pour créer une barre de menus de Mac OS, vous suivez les mêmes étapes que pour un Unix et la barre de menu Windows ci-dessus. Cela place un groupe de boutons, dont chaque propriété **menuMode** est réglée sur "**pulldown**", en haut de la fenêtre de votre pile.

Ensuite, vous définissez la propriété **menubar** de votre pile avec le nom du groupe.
Cela fait deux choses :
cela affiche les menus de la barre de menu en haut de l'écran, et cela raccourcit la fenêtre de la pile qu'il fait défiler, de sorte que le groupe de boutons de menu n'est pas visible dans la fenêtre.
Puisque les menus sont dans la barre de menu, vous n'avez pas besoin de les voir dans la fenêtre de la pile.

Important : Si votre pile a plus d'une carte, assurez-vous que le groupe est placé sur toutes les cartes. (Pour mettre un groupe sur une carte, choisissez le menu **Object > Place Group**, ou utilisez la commande **place**). Ceci garantit que la barre de menu sera accessible sur toutes les cartes de la pile, et empêche la pile de changer de taille lorsque vous passez d'une carte à l'autre.

La barre de menus par défaut

Si d'autres piles de votre application n'ont pas leurs propres barres de menu, définissez la propriété globale **defaultMenubar** au nom de votre groupe de menus, définissant ainsi la propriété **menubar** de la pile. Le **defaultMenubar** est utilisé pour n'importe quelle pile qui n'a pas une barre de menu qui lui est propre.

Astuce : Pour qu'une barre de menus personnalisée fonctionne correctement à l'intérieur de l'environnement de développement de LiveCode, vous devez définir **defaultMenubar** au nom de votre groupe de menus. Ceci remplace la barre de menu LiveCode IDE. Vous pouvez revenir à la barre de menu antérieure en choisissant l'outil pointeur.

Se Référer au Bouton Menu

Si le bouton est un bouton de menu qui est affiché dans la barre de menu, vous pouvez utiliser le mot **menu** de s'y référer :

```
get menuItem 2 of menu "Edit"
est identique à :
get line 2 of button "Edit"
```

Parce que les menus sont également des boutons, vous pouvez utiliser une référence de bouton pour obtenir

la même information. Mais vous pouvez avoir besoin de spécifier le groupe et la pile où le bouton se trouve pour éviter toute ambiguïté.

Par exemple, si il y a un bouton standard nommé **Edit** sur la carte actuelle, l'appel à ce bouton **Edit** se réfère à ce bouton, pas à celui de la barre de menu.

Une référence de bouton sans ambiguïté à un menu pourrait ressembler comme ceci :

```
get line 2 of button "Edit" of group "Menu" of stack "Main"
```

La déclaration ci-dessus produit les mêmes informations que la forme utilisant **menu**, mais vous avez besoin de connaître le nom du groupe et peut-être dans quelle pile il se trouve, donc le menu **menuNameform** est un peu plus pratique.

La couche de boutons de menu

Pour qu'une barre de menu, fonctionne parfaitement sur les systèmes OS X, les menus doivent être dans l'ordre des couches, au sein du groupe.

Ce qui implique que le bouton du menu **File** doit être numéroté en 1, le bouton du menu **Edition** doit être en 2, et ainsi de suite. **Menu Builder** prend en charge cela automatiquement ; vous ne devez vous soucier que de superposition si vous créez la barre de menu manuellement.

Changement de menus dynamique

Si vous souhaitez modifier dynamiquement le contenu d'un menu, avec un gestionnaire **mouseDown**, au moment où le menu est affiché, vous devez placer le gestionnaire **mouseDown** dans le le script du groupe. Quand un bouton de menu est affiché dans la barre de menus de Mac OS, il ne reçoit pas de messages **mouseDown**, mais son groupe oui.

La propriété editMenus

Lorsque vous définissez la propriété **menubar** d'une pile au nom d'un groupe, la pile est redimensionnée et remontée de sorte que la partie de la fenêtre qui contient les menus n'est pas visible.
Pour inverser cette action afin que vous puissiez la voir, et sélectionner, modifier les boutons qui composent la barre de menu, définissez la propriété **editMenus** à **true**.
Ceci redimensionne la fenêtre de la pile de sorte que les boutons des menus restent visibles, et vous pouvez utiliser les outils de l'environnement de développement de LiveCode pour y faire des changements.
Pour faire défiler la fenêtre de la pile à nouveau de sorte que les menus soient masqués, définissez la propriété **editMenus** à nouveau sur vrai.

Items de menu spéciaux

Quelques éléments de menu sur OS X sont traités directement par le système d'exploitation. Pour tenir compte de ces Items de menu spéciaux tout en vous permettant de créer une barre de menu entièrement multi-plateforme, LiveCode traite les deux derniers éléments de menu du menu d'aide (pour Mac OS X), dans le menu Fichier (OS X), et le menu Edit (OS X) de façon différente.

En suivant ces directives, vous pouvez vous assurer que vos menus s'affichent correctement sur tous les systèmes d'exploitation sans avoir à écrire de code spécial ou créer des barres de menus spécifiques à la plateforme.

Le menu Aide et le menu "A propos de cette application"

Quand LiveCode met en place la barre de menu de Mac OS, il crée automatiquement le dernier bouton du menu **Aide** (quel que soit le nom du bouton). Les éléments du menu d'aide standard, tels que "**A propos de cette application**" et "**Afficher les aides**" sur Mac OS Classic, sont inclus pour vous automatiquement, vous n'avez pas besoin de les inclure dans votre bouton du menu **Aide**, et vous ne pouvez pas les éliminer du menu **Aide**.
LiveCode déplace le dernier élément de menu du menu **Aide** vers la position **A propos de cette application**. Sur les systèmes Mac OS, c'est le premier élément de menu dans le menu **Pomme**. Sur les systèmes OS X, c'est le premier élément de menu dans le menu **Application**. Par conséquent, le dernier élément du menu dans votre bouton du menu **Aide** devrait être un élément approprié **A propos**. L'élément de menu supérieur doit être une ligne de séparation (un tiret), et au-dessus de ceci il doit y avoir au moins un élément du menu à placer dans le menu **Aide**.

Le menu Fichier et l'item de menu "Quitter"

Sur les systèmes OS X, l'item de menu "**Quitter**" est normalement placé dans le menu de l'application (qui est géré par le système d'exploitation) plutôt que dans le menu **Fichier**, comme c'est la norme sur d'autres plates-formes. Pour répondre à cette norme d'interface utilisateur, LiveCode supprime les deux derniers éléments de menu du menu **Fichier** lorsque une application autonome est exécutée sur un système OS X.

Par conséquent, le dernier élément du menu dans votre bouton de menu du fichier doit être **Quitter**. L'item de menu supérieur devrait être une ligne de séparation (un tiret).

Le menu Edition et l'item de menu «Préférences»

Sur les systèmes OS X, l'item de menu **Préférences** est aussi normalement placé dans le menu **Application**. Pour répondre à cette norme d'interface utilisateur, LiveCode supprime les deux derniers éléments de menu du menu **Edition** quand une application autonome s'exécute sur un système OS X. Par conséquent, le dernier élément du menu dans votre bouton de menu **Edition** devrait être "**Préférences**". L' item de menu supérieur devrait être une ligne de séparation (un tiret).

Note : L'élément de menu **Préférences** est traité de cette façon particulière seulement si son nom commence par la chaîne **Preferences**.

Astuce : Si l'interface utilisateur de votre application est présentée dans une langue autre que l'anglais, définissez le nom du bouton de menu Edition comme "**Edit**", et nommez son étiquette à votre traduction voulue. Cela garantit que le moteur peut trouver le menu Edition, tout en s'assurant que le menu est affiché dans la langue correcte.

Choisir les Items de menu spéciaux

Lorsque l'utilisateur choisit un des éléments d'un menu spécial, un message **menuPick** est envoyé au bouton, dont l'item du menu fait parti. Cela garantit que vos scripts de bouton fonctionneront sur toutes les plateformes, même si LiveCode affiche un élément de menu dans un menu différent pour se conformer aux directives d'interface utilisateur.

Par exemple, si l'utilisateur choisit **A propos de cette application** dans le menu **Pomme** sur un système Mac OS, un message **menuPick**, est envoyé sur le bouton **Aide** du menu, avec **A propos de cette application** comme paramètre. Vous gérez le message de l'élément de menu **A propos** dans le script du bouton du menu **Aide**, même si LiveCode affiche cette option dans un menu différent sur Mac.

7.13.7 Menus de pile

Les boutons menus peuvent être utilisés pour la plupart des types de menus standard. Toutefois, si vous voulez créer un menu avec une fonctionnalité qui n'est pas prise en charge par les menus boutons, par exemple, si vous voulez un menu contextuel qui fournit, comme type de choix, des images plutôt que du texte, vous pouvez créer un menu à partir d'une pile.

Création d'un menu pile

Pour créer un menu pile, vous créez une pile avec un contrôle pour chaque élément de menu. Puisque le menu pile est une pile et chaque élément du menu est un objet, les éléments de menu reçoivent des messages de la souris comme **mouseEnter**, **mouseLeave** et **mouseUp**.

Lorsque l'utilisateur choisit un élément dans le menu de la pile, un message **mouseUp** est envoyé à ce contrôle. Pour répondre à un choix d'item de menu, au lieu de traiter le message **menuPick**, vous pouvez placer un gestionnaire **mouseUp** dans le script de l'objet.

Pour créer un menu pile qui ressemble à un menu standard, créer un bouton dans la pile pour chaque élément de menu.
Les propriétés des boutons **autoArm** et **armBorder** devraient être mises sur vrai. Ou bien vous pouvez choisir l'élément "**Menu item**" dans le "**New Control**" sous-menu du menu **Object** pour créer un bouton avec ses propriétés définies par les valeurs appropriées.

Veillez à régler la propriété **rectangle** de la pile à la taille appropriée pour le menu. Rappelez-vous, lorsque vous ouvrez le menu, la pile sera affichée exactement comme dans une fenêtre modifiable.

Enfin, soit définir la propriété **menuName** d'un bouton pour créer une référence à la pile, soit placer un gestionnaire **mouseDown** contenant une commande **pulldown**, **popup** ou **option** dans le script d'un objet. Lorsque vous cliquez sur le bouton ou l'objet, le menu pile apparaît.

Affichage d'un menu pile

Les menus pile peuvent être associés à un bouton, tout comme les menus des boutons. Mais lorsque vous cliquez sur le bouton au lieu d'afficher un menu avec le contenu du bouton LiveCode affiche une pile avec le comportement d'un menu.

Vous pouvez également afficher un menupile sans l'associer à un bouton, en utilisant la commande **pulldown**, **popup** ou **option**. Normalement, vous utilisez ces commandes dans un gestionnaire **mouseDown**, de sorte que le menu apparaît sous le pointeur de la souris:

```
on mouseDown -- in card script
  popup stack "My Menu Panel"
end mouseDown
```

7.13.8 Affichage Menus contextuels

Il y a aussi plusieurs commandes pour afficher un menu contextuel. Habituellement vous utilisez ces commandes dans un gestionnaire **mouseDown** - normalement soit dans votre carte ou un script de pile

- commande **popup** : ouvre une pile comme un menu contextuel
- commande **pulldown** : ouvre une pile comme un menu déroulant
- commande **option** : ouvre une pile comme un menu d'options

Remarque : Si vous définissez la propriété **menuName** d'un bouton sur le nom d'une pile, le menu pile s'affiche automatiquement lorsque l'utilisateur clique sur le bouton. Vous devez utiliser les commandes **popup**, **pulldown** et **option** seulement si vous voulez afficher un menu pile d'une autre façon que par le clic sur un bouton.

7.14 Rechercher et Naviguer dans les cartes en utilisant la commande Find

La commande **find** de LiveCode vous permet de rechercher les champs de la pile courante, puis de naviguer et de mettre en évidence les résultats de la recherche automatique. Bien qu'il soit possible de construire une telle commande en utilisant les fonctions de comparaison, détaillées dans le chapitre ***Traitement du texte et des données***, dans la plupart des cas la commande **find** fournit une solution complète, pré-construite.

```
find [form] textToFind [in field]
find "heart"
find string "beat must go on" in field "Quotes"
```

Lorsque la commande **find** trouve une correspondance, elle met en évidence le résultat sur l'écran - si nécessaire en naviguant dans la carte qui contient le résultat, faisant défiler le champ de sorte que le texte soit visible.
La commande **find** peut également être utilisée pour renvoyer l'emplacement du texte qui a été trouvé. Pour réinitialiser la commande **find** afin qu'elle recommence la recherche au début :

```
find empty
```

Pour plus de détails sur la commande **find** et les options associées, voir la commande **find** dans le Dictionnaire de LiveCode.

7.15 Utiliser le glisser-déposer

LiveCode vous permet un contrôle complet sur le glisser-déposer - à la fois dans les fenêtres LiveCode, et entre LiveCode et d'autres applications.

7.15.1 Lancement d'un glisser-déposer

Pour commencer une opération de glisser-déposer, l'utilisateur clique et maintient le pointeur de la souris. Cela envoie un message **mouseDown** à l'objet.
Si vous faites glisser à l'intérieur d'un champ, un message **dragStart**, est envoyé. Pour permettre le glisser depuis un champ verrouillé ou depuis un autre type d'objet, dans le gestionnaire **mouseDown** de l'objet, définissez la propriété **dragData** en fonction du type de données que vous souhaitez faire glisser.
Quand une valeur est récupérée dans **dragData**, un drag and drop est déclenché lorsque la souris est cliquée, puis déplacée.

```
set the dragData["text"] to "text being dragged"
```

Vous pouvez définir **dragData** pour contenir l'un des types de données suivants:

text	Le texte brut est glissé.
HTML	Le texte stylé glissé dans le même format que **htmlText**
RTF	Le texte stylé glissé dans le même format que le **RTFText**
Unicode	Le texte glissé dans le même format que **UnicodeText**
image	Les données d'une image (au format PNG)
files	Le nom et l'emplacement du fichier ou des fichiers sont glissés, un par ligne

Note : LiveCode gère automatiquement le mécanisme de glisser-déposer de texte entre et dans les champs débloqués. Pour bénéficier ce type d'opération de glisser-déposer, vous n'avez pas besoin de faire de script.

Pour plus de détails, voir les entrées pour **dragStart** et **dragData** dans le Dictionnaire du LiveCode.

7.15.2 Suivi Pendant une Opération de Glisser-déposer

Vous pouvez utiliser le message **dragEnter** pour montrer un contour autour d'un objet ou de changer le curseur lorsque la souris se déplace dans le cours d'une opération de glisser.

```
on dragEnter            -- montre un contour vert autour de la cible
  set the borderColor of the target to "green"
end dragEnter
```

Vous pouvez utiliser le message **dragMove** pour rafraichir l'écran lorsque le curseur se déplace au cours d'une opération de glisser-déposer.

```
on dragMove             -- dans le script du champ modifie le curseur de sorte à ne pouvoir lancer qu'un lien effectif
  if the textStyle of the mouseChunk contains "link"
  then set the cursor to the ID of image "Drop Here"
  else set the cursor to the ID of image "Dont Drop"
end dragMove
```

Vous pouvez utiliser le message **dragLeave** pour enlever tout contour autour d'un objet ou changer le curseur lorsque la souris se déplace sur un objet pendant une opération de glisser.

```
on dragLeave            -- supprimer tout contour autour de l'objet qui n'est plus la cible
  set the borderColor of the target to empty
end dragLeave
```

Pour plus de détails, voir les entrées pour **dragEnter**, **dragMove** et **dragLeave** dans le Dictionnaire de LiveCode.

7.15.3 Répondre à un Drag and Drop

Pour effectuer une action lorsque l'utilisateur dépose des données sur un champ verrouillé ou un autre type d'objet, vous gérez le message **dragDrop**.
Le message **dragDrop** est envoyé lorsque l'utilisateur dépose des données sur un objet.

```
on dragDrop -- vérifier si un fichier est en cours de suppression
  if the dragData["files"] is empty then beep 2
  pass dragDrop
end dragDrop
```

Vous devez définir la propriété **acceptDrop** sur vrai avant qu'un déposer soit autorisé. Généralement, vous définissez cette propriété à vrai dans un gestionnaire **dragEnter**.

Vous pouvez utiliser la fonction **dragDestination** pour récupérer le long id de l'objet où les données glissées ont été déposées.

Vous pouvez utiliser la fonction **dragSource** pour récupérer le long id de l'objet qui est à l'origine du glisser.

Quand un glisser-déposer a été achevée, un message **dragEnd** est envoyé à l'objet où le drag and drop a débuté.

```
on dragEnd – supprime les données glissées
  delete the dragSource
end dragEnd
```

Vous pouvez utiliser la fonction **dropChunk** pour récupérer l'emplacement du texte qui a été déposé dans un champ. Par exemple, vous pouvez sélectionner le texte qui a été déposé en procédant comme suit :

```
select the dropChunk
```

Pour plus de détails, voir les entrées pour **dragDrop**, **dragEnter**, **dragDestination**, **dragEnd**, **dragSource**, **dropChunk** et **acceptDrop** dans le Dictionnaire du LiveCode.

7.15.4 Empêcher glisser-déposer sur un champ

Vous empêchez de déposer des données dans un champ pendant un drag and drop en définissant la propriété **acceptDrop** à faux lorsque le pointeur de la souris entre dans le champ.

Si le **acceptDrop** est réglé sur faux, lorsque vous déposez les données, aucun message **dragDrop** n'est envoyé sur ce champ.

Puisque le dépôt est traité automatiquement seulement quand LiveCode reçoit un message dragDrop, cela empêche le comportement habituel de dépôt automatique.

Habituellement, vous devez définir **acceptDrop** dans un gestionnaire **dragEnter**, comme dans l'exemple suivant :

```
on dragEnter        -- dans le script du champ
  set the acceptDrop to false
end dragEnter
```

Si vous voulez empêcher le glisser du texte dans un champ, intercepter le message dragStart :

```
on dragStart        -- ne rien mettre pour piéger le message
end dragStart
```

Pour plus de détails, consultez les entrées pour **acceptDrop**, **dragDrop** et **dragEnter** dans le Dictionnaire du LiveCode.

Chapitre 8 Bases de Données et Impression

A. Bases de Données

Avec la bibliothèque **Database** de LiveCode, votre application peut communiquer avec les bases de données **SQL** externes.
Vous pouvez obtenir des données de bases mono-utilisateur et multi-utilisateur, faire des mises à jour des données, obtenir des informations sur la structure de la base de données et afficher les données de la base de données dans votre pile. Et avec le Générateur de requêtes de base de données, vous pouvez automatiser le processus d'interrogation d'une base de données et remplir les champs avec les données, sans les scripts nécessaires. Pour une discussion pour savoir quand il est approprié d'utiliser une base de données externe avec LiveCode, consultez la rubrique ***Quand utiliser une base de données*** dans le chapitre 2.

Ce chapitre explique comment installer le logiciel nécessaire pour communiquer avec les bases de données, comment mettre en place des requêtes de base de données automatiques à l'aide du **Database Query Builder** (Générateur de requêtes de base de données), et la façon d'utiliser la bibliothèque de base de données pour communiquer entre LiveCode et une base de données .

Cette rubrique ne comprend pas les discussions sur la façon de configurer et créer une base de données SQL, ce qui est au-delà de la portée de la documentation LiveCode.

Pour bien comprendre cette rubrique, vous devez savoir comment écrire des scripts courts et devez comprendre les concepts de base de données SQL (lignes et colonnes, les curseurs de base de données et des requêtes SQL).

Quelques termes utilisés dans ce sujet, comme «champ» et « curseur », font partie de la terminologie standard pour travailler avec des bases de données, mais ont un sens différent dans le contexte de développement de LiveCode.
Lors d'un va-et-vient entre le travail sur base de données et le développement d'une application plus générale, assurez-vous de comprendre quelle acception est applicable dans le contexte dans lequel vous travaillez actuellement.
Lorsque vous faites référence à des termes spécifiques à la base de données, la documentation utilise généralement des phrases comme "champ de base de données" ou "curseur de la base de données" pour vous rappeler le contexte.
Voir le glossaire à l'intérieur de la documentation du produit pour les définitions de tout terme dont vous n'êtes pas certain.

8.1 Introduction à l'accès aux bases de données

Une base de données est une ressource externe qui contient des informations, structurées sous une forme spéciale pour un accès rapide et extraction de données. Les bases de données peuvent être :

- de toute taille, de petite à très grande,
- situé sur le même système que l'application ou sur un serveur distant,
- accessible par un seul utilisateur à la fois ou par plusieurs utilisateurs à la fois

8.1.1 Bases de données SQL

Une base de données **SQL** vous permet d'accéder et de contrôler vos données, via **SQL**, un langage de base de données d'accès standard et largement utilisé. Vous utilisez des requêtes **SQL** (déclarations dans le langage **SQL**) pour spécifier la partie de la base de données avec laquelle vous voulez travailler, pour obtenir des données, ou apporter des modifications à la base de données.

L'accès aux bases depuis LiveCode est complet. Vous pouvez envoyer toute instruction **SQL** à une base de données. Vous pouvez ouvrir plusieurs bases de données (ou plusieurs connexions à la même base de données), maintenir plusieurs jeux d'enregistrement (curseurs de base de données) par connexion, et envoyer et recevoir des données binaires ainsi que du texte. Vous pouvez faire tout cela en utilisant **Database Query Builder**, ou dans des scripts qui utilisent les commandes et fonctions de la bibliothèque **Database**.

Pour voir une liste de termes de LiveCode dans la bibliothèque **database**, ouvrez le dictionnaire, et tapez **database** dans le champ de filtre de recherche.
Vous pouvez aussi trouver un atelier interactif et de tutoriels sur notre site Web à l'adresse:
http://lessons.runrev.com/s/lessons/m/4071/c/16767

8.1.2 Pourquoi utiliser une base de données externe ?

Voir la section ***Quand utiliser une base de données*** dans le chapitre 2.

8.1.3 Les bases de la structure de la base de données

Une base de données est constituée d'enregistrements, qui à leur tour sont restitués depuis des champs dans la base de données. Un champ est la plus petite partie d'une base de données qui puisse être abordée séparément. Chaque champ de bases de données contient un type particulier d'information. Ce peut-être un nom, un chemin de fichier, une image, ou tout autre type d'information. Chaque enregistrement contient une valeur pour chacun de ses champs. Un ensemble d'enregistrements est appelé une table de bases de données, et une ou plusieurs tables, constituent une base de données.

Voici un exemple : supposons que vous ayez une base de données de clients pour votre entreprise. Les champs de cette base de données peuvent inclure le nom du client, un numéro d'identification unique du client, et l'adresse d'expédition. Chaque enregistrement comprend les informations relatives à un client unique, de sorte que chaque enregistrement a un autre nom du client, adresse de livraison, etc.

Remarque : Vous avez sans doute remarqué que la structure de la base de données telle que décrite ressemble à une pile multi-carte qui a les mêmes champs sur chaque carte. Un champ de bases de données est comme un champ dans une pile, et un enregistrement est comme une carte. Une pile configurée de cette manière peut agir comme une base de données, en fait, mais il lui manque certaines des caractéristiques d'une base de données externe, comme la capacité d'exécuter des requêtes **SQL** et la capacité de rester robuste pour les accès multi-utilisateurs.

Vous pouvez aussi penser à l'ensemble des enregistrements clients sous forme de grille (comme un tableur). Chaque ligne est un enregistrement et chaque colonne est un champ, de sorte que chaque cellule de la grille contient un autre élément d'information au sujet d'un client particulier. Voici un exemple :

ID	Nom_Client	Adresse	Pays
123	Jane Jones	234 E. Street	U.K.
836	Acme Corporation	PO Box 23788	USA
823	CanCo, Inc.	1 CanCo Blvd.	Japan

Exemple de grille de base de données

Il y a trois lignes dans cette grille (chacun est l'enregistrement d'un client particulier) et quatre colonnes (chacun est l'un des champs).
Une ligne de la base de données signifie un enregistrement de client unique, qui a une valeur pour chaque champ. Une colonne de la base de données signifie de l'ensemble de toutes les valeurs de l'un des champs, un pour chaque enregistrement (par exemple, l'ensemble de toutes les adresses de clients). Plus généralement, chaque ligne décrit l'une des choses dans la base de données, et chaque colonne décrit un état particulier de chaque chose dans la base de données.
L'ensemble des enregistrements clients constitue une table. Votre base de données peut inclure seulement cette table, ou elle peut inclure d'autres tables connexes, comme une liste de toutes les ventes. Vous pouvez également connecter des données connexes provenant de différentes tables de la même base de données. Par exemple, si chaque enregistrement de latable Ventes inclut le numéro d'identification du client qui a acheté le produit, vous pouvez relier tous les enregistrements de ventes pour un client à l'enregistrement de ce client dans latable Clients.

8.1.4 SQL et Enregistrements - Curseurs de base de données

SQL travaille principalement avec des des jeux d'enregistrements, plutôt que des lignes individuelles. Lorsque vous envoyez une requête SQL à une base de données, la requête sélectionne généralement certains enregistrements où vous pouvez y effectuer d'autres opérations. L'ensemble des enregistrements résultant d'une requête SQL est appelé un curseur de la base de données, ou une série d'enregistrements. Les

requêtes SQL vous permettent de décrire les caractéristiques des enregistrements dont vous avez besoin, au lieu de traiter chaque enregistrement un par un pour savoir s'il correspond à vos critères.

Par exemple, considérons latable des clients, nous avons parlé dans la section précédente. Pour travailler uniquement avec des clients américains, vous pouvez écrire une requête SQL qui sélectionne uniquement les enregistrements où le champ pays est "USA". Ce sous-ensemble d'enregistrements devient alors un jeu d'enregistrements.
Vous pouvez en savoir plus sur les champs et les enregistrements contenus dans ce jeu d'enregistrements, et passer d'un enregistrement à l'autre sein de ce sous-ensemble de la base de données. Vous pouvez créer plus d'un jeu d'enregistrements pour travailler avec plus d'un jeu d'enregistrements à la fois.

Remarque : les implémentations de bases de données ont différentes limitations sur les mouvements dans un jeu d'enregistrements. Par exemple, certaines bases de données ne vous laisseront pas aller en arrière dans un jeu d'enregistrements : à la place, vous devez commencer par le premier enregistrement et avancer afin d'examiner les données.

8.1.5 Choix d'une base de données

LiveCode prend directement en charge les implémentations des bases de données suivantes :

- Oracle
- MySQL
- SQLite
- PostgreSQL
- Valentina

LiveCode prend également en charge la connexion à une base de données via **ODBC**. Vous pouvez utiliser **ODBC** pour utiliser **Access**, **FileMaker**, **MS SQL Server** et beaucoup d'autres implémentations de bases de données. Voir ci-dessous pour plus d'informations sur **ODBC**.

Les commandes et les fonctions de bases de données depuis LiveCode utilisent la même syntaxe indépendamment du type de base de données à laquelle vous vous connectez. Vous n'avez pas besoin d'apprendre un langage de base de données distinct pour chaque type. Au lieu de cela, lorsque vous ouvrez une base de données avec la fonction **revOpenDatabase**, vous spécifiez son type comme l'un des paramètres de sorte que LiveCode sache à quel type de base de données il a à faire. La bibliothèque **Database** gère les détails pour chaque type dans les coulisses pour vous.

Lorsque vous utilisez **Database Query Builder**, il vous suffit de sélectionner le type de base de données que vous souhaitez utiliser. Comme la bibliothèque de base de données, **Database Query Builder** gère les détails pour vous. Vous pouvez facilement basculer entre les types de bases de données.

8.2 Raisons du Choix d'un Type de Bases de données

Choisir tel type de base de données, dépend d'un certain nombre de facteurs. Si vous avez besoin de travailler avec une base de données existante, ou qu'un gestionnaire d'une base de données est déjà installé, la décision s'impose à vous.

De même, si vous êtes déjà un expert de la mise en oeuvre d'une base de données particulière, vous préférerez probablement continuer utiliser celle-ci.

D'autres facteurs dans votre choix peuvent inclure prix, performances, modèle de licence (source commerciale ou open source), et le support de la plate-forme. Si vos utilisateurs sont sur un ensemble particulier de plates-formes, vous devriez choisir une base de données qui est prise en charge sur toutes les plateformes. **Runtime** ne cautionne pas un quelconque type de base de données particulier, les informations fournies sur les différents types le sont à titre informatif seulement.

8.2.1 Vue d'ensemble d' ODBC

Open Database Connectivity (**ODBC**) est un système qui permet aux développeurs d'accéder à tout type de base de données compatible de manière standard.

Pour communiquer avec une base de données, vous avez généralement à ajouter du code qui utilise des protocoles propriétaires de la base de données. Sans **ODBC**, afin de créer un programme qui puisse communiquer avec, par exemple, **FileMaker**, **Access** et des bases de données **Oracle**, LiveCode devra inclure le code pour trois protocoles de bases de données différents. Avec **ODBC**, LiveCode peut communiquer avec chacun de ces types de bases de données en utilisant le même code.

Les gestionnaires ODBC

Pour travailler avec des bases de données via **ODBC**, vous avez besoin de deux logiciels : un gestionnaire **ODBC**, plus un pilote de base de données pour le type de base de données spécifique que vous utilisez.

Windows 2000, Windows XP, et Mac OS X version 10.2 et versions ultérieures, incluent le logiciel **ODBC** avec le système d'exploitation. Pour les versions antérieures, et pour les systèmes Unix, vous pouvez télécharger un gestionnaire **ODBC** et un ensemble de pilotes. (Voir la section ci-dessous intitulée «Software pour l'accès aux bases de données" pour plus d'informations.)

8.2.2 Performance entre un Accès Direct et un Accès via ODBC

En règle générale, l'accès à une base de données via **ODBC** demande plus de configuration et est plus lent que l'accès direct à la base de données. Pour cette raison, LiveCode offre la possibilité d'accéder à MySQL, PostgreSQL et Oracle directement sans passer par le protocole **ODBC**. Cette capacité sera précieuse pour tous ceux qui font un travail de base de données professionnelle complexe ou étendue.
La syntaxe des fonctions de la bibliothèque **Database** est identique pour tous les types de bases de données, de sorte que vous n'avez pas besoin de réécrire vos scripts pour profiter de l'efficacité accrue de l'accès direct.

8.3 Logiciel pour Accéder Aux Bases de Données

Pour fournir une connectivité aux bases de données, LiveCode fonctionne avec des pilotes de bases de données (logiciel qui convertit les demandes d'application dans le protocole requis par une base de données spécifique).

8.3.1 Trouver des Pilotes de Bases de Données

Les pilotes de base de données pour certains types de bases de données sont inclus dans la distribution de LiveCode. La liste des types de bases de données incluses dépend de la plate-forme. Les sections ci-dessous répertorient les pilotes de bases de données comprises et avec quelle plateforme.
Si vous avez installé LiveCode, vous avez tout les logiciels nécessaires pour utiliser tous les types de bases de données inclus. Pour les autres types de bases de données, vous aurez besoin d'obtenir les pilotes de bases de données appropriées avant de pouvoir travailler avec les bases de données.

Important : Cette section contient des liens vers des sites Web tiers et contient des informations sur des logiciels tiers. Cette information est fournie pour votre confort, mais RunRev n'est pas responsable des logiciels et des sites référencés. Runtime regrette de ne pouvoir fournir aucune aide pour l'installation et la configuration d'une base de données.

8.3.2 MySQL

Les pilotes de bases de données MySQL sont inclus dans le cadre de l'installation de LiveCode Linux, Mac OS X et Windows.

8.3.3 Oracle

LiveCode inclut des pilotes pour Oracle sur Windows et (actuellement) PPC Mac OS X. Pour obtenir un pilote de base de données Oracle pour votre plate-forme, visitez le site Web d'Oracle :
http://www.oracle.com.

8.3.4 PostgreSQL

Un pilote de base de données PostgreSQL est inclus dans le cadre de l'installation de LiveCode Linux, Mac OS X et Windows.

8.3.5 SQLite

Des pilotes pour accéder à cette base de données sont inclus avec LiveCode. Aucune installation supplémentaire n'est nécessaire.

8.3.6 Gestionnaires ODBC et Pilotes de Bases de Données

Pour utiliser une base de données via **ODBC**, vous devez installer le logiciel **ODBC** nécessaire à votre plate-forme. (Certains systèmes d'exploitation incluent une installation **ODBC**). Le logiciel **ODBC** inclut un ou plusieurs pilotes de bases de données, plus un utilitaire de gestion **ODBC**.

ODBC sur système Windows

Windows 2000 et Windows XP incluent le package **MDAC** (Microsoft Data Access Components) dans le cadre de l'installation du système standard. Pour configurer **ODBC** sur les systèmes Windows, utilisez le panneau de contrôle des sources de données **ODBC**.

ODBC sur système Mac OS X

OS X version 10.2 jusqu'à 10.5 contient un logiciel **ODBC** dans le cadre de l'installation du système standard. Pour configurer **ODBC** sur les systèmes OS X, utilisez l'application **Administrateur ODBC** dans le dossier Utilitaires.
Depuis 10.5, **ODBC** n'est plus présent dans le système. Pour l'installer, aller à cette adresse :
http://www.iodbc.org/dataspace/iodbc/wiki/iODBC/
Egalement on peut télécharger **ODBC Manager** :
http://www.odbcmanager.net/

ODBC sur système Unix

LiveCode soutient **ODBC** et **UnixODBC** sur les systèmes Unix. Vous pouvez télécharger le logiciel **ODBC** à partir du site web **ODBC** : http://www.iodbc.org/.
Vous pouvez télécharger le le logiciel **unixODBC** à partir du site web **unixODBC** à http://www.unixodbc.org.

Création d'un DSN pour l'accès ODBC

Une fois que vous avez installé le logiciel nécessaire, vous utilisez le gestionnaire **ODBC** pour créer un **DSN** (Data Source Name), qui est une spécification qui définit une base de données particulière. Vous utilisez le **DSN** pour vous connecter à la base de données via **ODBC**. L'exemple suivant ouvre une connexion à une base de données dont le **DSN** est nommé **myDB** :

```
get revOpenDatabase("ODBC","myDB",,"jones","pass")
```

Pour se connecter à une base de données en utilisant le **Database Query Builder**, vous entrez le nom du **DSN** dans la fenêtre Générateur de requêtes de base de données.

Note: L'un des avantages de la mise en place d'un **DSN** est que si vous souhaitez modifier l'emplacement de la base de données, il suffit de modifier les paramètres **DSN**, pas votre code d'application. Vous pouvez penser à un **DSN** comme une sorte de raccourci ou alias de votre base de données.

B. Impression et Rapports

L'impression est un aspect vital de nombreuses applications. LiveCode fournit un ensemble complet de fonctionnalités d'impression. Si vous voulez une copie simple de votre pile, souhaitez imprimer des étiquettes ou des rapports complexes, LiveCode présente les caractéristiques dont vous avez besoin.

8.4 Introduction à l'impression

LiveCode prend en charge un certain nombre de méthodes d'impression. Vous pouvez utiliser la commande **print card** et avoir LiveCode qui gère la mise en page des cartes sur le papier.
Sinon, vous pouvez utiliser les commandes de **print into rectangle** qui vous permettent de prendre le plein contrôle de la mise en page de votre impression. La première méthode est plus adaptée à la mise en œuvre des fonctionnalités d'impression simple, tandis que la seconde est mieux adaptée pour l'impression de mise en page plus complexe ou l'impression des rapports. Enfin, vous pouvez utiliser la bibliothèque de scripts d'impression de champ prédéfinie pour imprimer le contenu d'un champ de texte à l'aide d'une simple commande.

LiveCode comprend également un ensemble complet de fonctionnalités d'accès et de réglage des options du périphérique d'impression, y compris des options telles que les marges, l'étendue de pages et le nombre de copies.
Sur les systèmes Mac OS X, vous pouvez également imprimer directement sur un fichier PDF - même sans montrer à l'utilisateur une boîte de dialogue d'impression.
Sur les systèmes Windows, cette option permet de créer un fichier XPS et sur les systèmes Unix un fichier Postscript.
Cette fonctionnalité est inestimable si vous voulez, respectivement, produire un fichier PDF haute résolution, XPS ou un fichier Postscript, à partir de votre pile.

8.5 Contrôle des paramètres de l'appareil de l'imprimante

LiveCode vous donne un contrôle complet par programmation sur votre périphérique d'impression ainsi que les paramètres disponibles.

8.5.1 Choix d'une imprimante

Utilisez **availablePrinters** pour lister les imprimantes disponibles sur le système de l'utilisateur. Les imprimantes peuvent inclure les modems, le fax et les dispositifs en réseau. Si les **availablePrinters** est vide, aucune imprimante n'est affectée. Par exemple, pour placer une liste des imprimantes disponibles dans un champ de liste :

```
put the availablePrinters into field "list of printers"
```

Réglez le **printerName** sur l'imprimante à utiliser pour imprimer. Vous pouvez utiliser n'importe quelle imprimante figurant dans **availablePrinters**. Cette propriété est utile lors de la production d'un utilitaire maison qui a besoin d'imprimer sur une imprimante spécifique sur le réseau de l'entreprise, ou pour restaurer automatiquement le précédent choix d'imprimante de l'utilisateur stocké dans un fichier de préférences.

```
set the printerName to the cSavedPrinter of stack "My Preferences"
```

printerFeatures fournit une liste des fonctionnalités supportées par l'imprimante sélectionnée. Ces

caractéristiques varient largement d'un appareil à l'autre, mais les caractéristiques typiques peuvent inclure des choses telles que "Copies assemblées", "Couleur" ainsi que "Duplex (Recto-Verso)". Utilisez cette propriété pour activer et désactiver les options de sortie dans toute boite de dialogue dédiée aux paramètres personnalisés de votre imprimante.

8.5.2 Choisir le Mode de Sortie (par exemple Imprimer Dans un Fichier)

La propriété globale **PrinterOutput** vous permet de choisir le mode de sortie pour les commandes d'impression ultérieures. Cette propriété est définie par le système pour l'imprimante par défaut au démarrage et sera changée si vous ouvrez une boîte de dialogue d'impression du système dans lequel l'utilisateur peut sélectionner une autre imprimante.
Si cette propriété est définie sur le périphérique, elle sera envoyée à l'imprimante physique.

Alternativement, vous pouvez le régler sur un chemin de fichier pour imprimer dans un fichier. Sur Mac OS X cela va créer un fichier PDF, sur Windows un fichier XPS ainsi que sur UNIX un fichier postscript.
Sur Mac OS X, vous pouvez le définir sur Aperçu pour créer un aperçu.
Par exemple, pour enregistrer la carte actuelle dans un fichier :

```
ask file "Save as:"
set the printerOutput to ("file:" & it )
print this card
```

8.5.3 Travailler avec des Dialogues d'Impression

Dans la plupart des applications qui ont besoin d'imprimer, vous devrez fournir un moyen de mettre en place les boîtes de dialogue standard du réglage de la page d'impression de l'OS. En général, vous souhaitez rendre ces options disponibles à partir de la page d'impression ainsi que les éléments de configuration dans le menu **Fichier** de votre application. Lorsque l'utilisateur effectue des modifications dans ces boîtes de dialogue, les modifications vous sont rendues accessibles dans les propriétés globales. Nous abordons ici comment sauvegarder et utiliser ces propriétés ci-dessous.

Remarque : Il n'est pas nécessaire de mettre en place les dialogues de l'OS standard pour modifier les paramètres de l'imprimante. Vous pouvez définir les paramètres d'impression appropriés directement par un script à la place.

Remarque : Sur les systèmes Linux et Unix, vous aurez besoin d'une version récente de GTK installée afin d'afficher la boîte de dialogue d'imprimante de l'OS. Si vous ne l'avez pas installé, LiveCode affiche sa propre boîte de dialogue d'imprimante qui a été construite comme une pile et une bibliothèque de script. Cette boîte de dialogue imite la boîte de dialogue d'imprimante standard du système et fixe les les propriétés globales d'impression de LiveCode, directement.

Astuce : Vous pouvez forcer LiveCode à utiliser son propre dialogue des paramètres d'impression interne en définissant la propriété globale **systemPrintSelector** sur faux. Les utilisateurs avancés peuvent personnaliser la boîte de dialogue d'impression qu'utilise LiveCode en exécutant **toplevel "print dialog"** ou **toplevel "page setup"** dans la boîte de message. N'oubliez pas de sauvegarder une copie de la pile car elle sera écrasée à chaque fois que vous mettez à jour votre copie de LiveCode.

Important : Les configurations des boîtes de dialogue d'impression et de la page de LiveCode doivent être incluses dans une application autonome si vous les utilisez. Assurez-vous que la case à cocher **Print Dialog** est activée dans le dialogue **Standalone Application Settings** de votre application. Vous n'avez pas besoin d'inclure ces dialogues si vous utilisez uniquement les dialogues d'impression natifs de l'OS. Pour plus d'informations sur l'intégration des ressources dans votre application autonome, voir le chapitre sur la création d'applications autonomes

Pour faire apparaître la boîte de dialogue standard de l'imprimante de l'OS, utilisez la commande **answer printer** :

```
answer printer
```

Si l'utilisateur n'appuie pas sur le bouton Annuler alors les modifications apportées aux paramètres de l'imprimante seront reflétées dans les propriétés d'impression globaux, discutés ci-dessous.

Pour faire apparaître la boîte de dialogue de configuration de la page de l'OS standard, utilisez la commande **answer page setup**.

```
answer page setup
```

8.5.4 Sauvegarde des paramètres de l'imprimante

Pour enregistrer ou de définir un ensemble complet d'options relatives à l'imprimante active, qui intègre tous les paramètres dans la mise en page et d'impression de l'OS dialogues, utilisez la propriété globale **PrinterSettings**.

La propriété **PrinterSettings** est une chaîne binaire qui décrit complètement les paramètres actuels. La propriété contient le nom de l'imprimante et les paramètres actuellement utilisés.

Attention : Vous ne devriez pas tenter de modifier les **PrinterSettings** mais plutôt obtenir et les définir dans leur intégralité. Pour accéder aux propriétés d'impression individuelles, utilisez les propriétés d'impression globale décrite ci-dessous.

Quand vous définissez la propriété **PrinterSettings** à une valeur enregistrée, LiveCode va choisir l'imprimante nommée dans la propriété et régler tous ses paramètres à celles contenues dans cette propriété. Si l'imprimante ne peut être trouvée LiveCode renvoie l'erreur "**unknown printer**" dans le résultat. Si l'imprimante est détectée mais les paramètres ne sont pas valides alors LiveCode va choisir l'imprimante et la réinitialiser aux valeurs par défaut.

Remarque : Vous devez enregistrer une copie distincte de la propriété **PrinterSettings** pour chaque imprimante ou OS que vous comptez utiliser. La propriété **PrinterSettings** peut pas être transférée entre les plates-formes. Par exemple, une propriété **PrinterSettings** générée sur un ordinateur Windows ne peut pas être utilisée sur Mac OS X - pour une même imprimante. Si vous voulez modifier les paramètres entre les différentes plates-formes et types d'imprimantes, utilisez les propriétés d'impression globales décrites ci-dessous. Utilisez **PrinterSettings** pour plus de commodité lorsque vous savez que vous allez utiliser la même imprimante et que vous voulez enregistrer tous les paramètres, ou lorsque vous définissez des propriétés ésotériques qui ne figurent pas dans les propriétés d'impression globales décrites ci-dessous.

Pour enregistrer les paramètres actuels de l'imprimante dans une propriété personnalisée stockée sur la pile actuelle :

```
set the cSavedPrinterSettings of this stack to the printerSettings
save this stack
```

Ensuite, pour restaurer ces paramètres :

```
set the printerSettings to the cSavedPrinterSettings of this stack
```

8.6 Papier Options Connexes

Cette section explique comment vous obtenez et configurez les options liées au papier - la zone de rectangle de papier, le format du papier, l'orientation et l'échelle à utiliser pour imprimer sur le papier. Ces options de papier s'appliquent à tous les types d'impression, à savoir la carte, le champ, et la mise en page d'impression.

Utilisez le **printRectangle** pour obtenir la surface de rectangle imprimable dans le papier (valeurs retournées en coordonnées de l'appareil). Cette propriété prend en compte tous les paramètres appliqués par l'utilisateur dans la mise en page ainsi que les boîtes de dialogue d'impression, y compris l'orientation d'impression (par exemple paysage ou portrait). Le rectangle est représenté à gauche, haut, droite, bas et est toujours relatif à la partie supérieure gauche de la page - donc en haut à gauche sera toujours 0,0. Le **printRectangle** sera toujours à l'intérieur de la **printPaperRectangle** - la surface rectangulaire de la feuille de papier.

Remarque : La propriété **printRectangle** est en lecture seule et ne peut pas être réglée directement. Pour modifier cela, vous devez définir d'autres options relatives au papier, par exemple le **printPaperOrientation** (voir ci-dessous).

Important : Ne confondez pas **printMargins** et d'autres propriétés d'impression de mise en page des cartes avec les propriétés du papier telles que **printRectangle**. **printMargins** ne s'applique qu'à l'impression des cartes, en utilisant les capacités de mise en page automatiques des cartes depuis LiveCode (voir ci-dessous). Ainsi, **printMargins** n'a aucun effet sur **printRectangle**.

Utilisez le **printPaperOrientation** pour obtenir et définir l'orientation de votre impression. Cette propriété peut être définie à l'une des valeurs suivantes :

- ***portrait*** : rotation 0 degrés.
- ***Landscape*** : rotation 90 degrés horaire.
- ***reverse portrait*** : rotation 180 degrés anti-horaire.
- ***reverse landscape*** : 270 degrés horaire.

```
set the printPaperOrientation to "landscape"
```

Utilisez la propriété **printPaperScale** pour appliquer un facteur d'échelle pour votre impression après que tous les autres paramètres aient été pris en compte.

Remarque : printPaperScale est appliqué après tout autre disposition et options de mise à l'échelle. Par exemple, si vous avez utilisé les fonctions d'impression de mise en page pour imprimer une série de cartes à l'échelle de 50%, puis réglez le **printPaperScale**, ce facteur sera appliqué à la mise en page entière après que la mise à l'échelle de la carte ait été calculée.

Pour imprimer dans une plage comprise entre 1 et 100%, réglez le **printPaperScale** sur un nombre compris entre 0 et 1.
Pour imprimer à 200%, réglez le **printPaperScale** à 2.

```
set the printPaperScale to 0.5 -- 50%
```

8.7 Tâches Options connexes

Cette section explique comment obtenir et définir les options liées aux tâches - le nombre de copies, l'impression recto verso, la collation, la couleur, le titre et zone d'impression.

Important : Les options d'emploi disponibles dépendent toutes de ce que l'imprimante sélectionnée prend en charge (utilisez la propriété **printerFeatures**, décrite ci-dessus, pour récupérer une liste des fonctions prises en charge par l'imprimante).

Utilisez la propriété **printCopies** pour obtenir et définir le nombre de copies à imprimer. **printCopies** devrait être fixé à une valeur de 1 ou plus.

```
set the printCopies to 5 -- imprimer 5 exemplaires
```

Utilisez la propriété **printDuplex** pour indiquer à l'imprimante d'imprimer recto-verso. Cette propriété peut être définie à l'une des valeurs suivantes :

- **none** : pas d'impression recto-verso
- **short edge** : impression recto-verso bord court
- **long edge** : impression recto-verso bord long.

```
set the printDuplex to "short edge"
```

Utilisez la propriété **printCollate** pour spécifier s'il faut ou pas entrelacer plusieurs copies d'un travail d'impression. Si un travail d'impression comporte trois pages, P1, P2 et P3 :

- avec **printCollate** définie sur vrai et **printCopies** fixé à 2 l'ordre de sortie sera P1, P2, P3, P1, P2, P3.
- avec **printCollate** définie sur faux, la sortie sera P1, P1, P2, P2, P3, P3.

```
set the printCollate to true
```

Utilisez la propriété **PrintColors** pour spécifier s'il faut imprimer en couleur ou non. Si "couleur" n'est pas dans les paramètres remontés dans **printerFeatures** cette propriété n'aura aucun effet et tous les travaux d'impression seront imprimés en noir et blanc. Cette propriété peut être soit réglée sur vrai, soit sur faux. Par exemple, pour vérifier si l'impression couleur est supportée sur l'imprimante utilisée et s 'en servir si possible :

```
if "color" is among the lines of the printerFeatures then
   set the printColors to true
end if
```

Utilisez la propriété **PrintTitle** pour spécifier le nom de la prochaine tâche d'impression dans la file d'attente de l'imprimante. La définition de cette propriété permet de connaître le nom du document de l'utilisateur, de sorte que l'utilisateur soit en mesure de le reconnaître dans l'utilitaire de file d'attente d'imprimante. Si le **PrintTitle** est vide au début d'une boucle de l'impression, le titre de la pile par défaut, **defaultStack**, sera utilisé.

```
set the printTitle to "My Report 1"
```

Utilisez la propriété **printRectangle** pour déterminer la zone imprimable de la page physique telle qu'elle est indiquée par l'imprimante.
Ce rectangle sera toujours contenu dans le **printPaperRectangle**.
Ainsi, vous devriez utiliser le **printRectangle** et non le **printPaperRectangle** lors du calcul de mise en page. **printPaperRectangle** est utile si vous générez un aperçu avant impression et voulez afficher toute la surface du papier, y compris les zones de marge qui ne peuvent être imprimées. Cette propriété est en lecture seule et ne peut pas être réglée directement.

8.7.1 Imprimante Métrique des Polices (Windows)

Les systèmes Windows utilisent parfois des versions différentes de la même police pour afficher du texte à l'écran et l'impression. Cela peut entraîner des mises en page et les sauts de ligne différentes entre l'écran et la sortie imprimée.
Pour éviter cela, vous pouvez dire à LiveCode d'utiliser les polices d'imprimante pour affichage sur écran. Pour ce faire, définissez la propriété des piles, **formatForPrinting**, à true.

Faire et ne pas faire : imprimante - métriques de police sous Windows

Faire :	Ne Pas Faire :
Définir la propriété de la pile **formatForPrinting** sur vrai avant de charger une pile en mémoire. Si la pile est déjà chargée, définissez cette propriété sur vrai, puis enregistrez et rechargez-la. Enregistrer alors en utilisant **Close et Remove from memory** dans le menu Fichier). Créer une pile hors écran avec **formatForPrinting** défini sur vrai avec votre modèle de mise en page d'impression et de copie du texte insérés avant l'impression. Régler le **formatForPrinting** avant de faire des calculs liés à la mise en page d'impression sur la pile. Règler le **formatForPrinting** sur vrai sur n'importe quelle pile d'aperçu avant impression qui s'affiche à l'utilisateur.	Permettre à l'utilisateur d'éditer le texte directement dans les champs dont la propriété **formatForPrinting**, est réglée sur vrai. Tenter de le faire peut causer des anomalies d'affichage. Définissez cette propriété sur faux et recharger la pile en premier. Utiliser généralement des des piles, avec **formatForPrinting**, mis sur vrai pour un affichage sur écran, car cela va afficher le texte qui a été optimisé pour l'affichage d'impression (au lieu de l'affichage de l'écran, non adapté à la lecture sur l'écran). Utiliser cette propriété sur d'autres plates-formes. Windows est la seule plate-forme qui utilise différentes polices à l'écran par rapport à l'impression. Utiliser la propriété **windowBoundingRect** pour limiter l'affichage d'une pile dont **formatForPrinting**, a été fixé sur vrai - cette propriété sera ignorée lorsque la pile est ouverte ou maximisée.

8.8 Imprimer Une Carte

Une fois que vous avez configuré votre imprimante, réglé les options de papier et de tâches (voir ci-dessus), vous êtes maintenant prêt à utiliser l'une des commandes d'impression pour lancer l'impression.
Au plus simple, la commande d'impression de la carte vous permet d'imprimer une carte. Plus tard, nous allons aborder les possibilités d'imprimer des mises en page plus complexes, champs et texte.

```
print this card -- imprime la carte actuelle
print card 12 -- imprime la carte 12
```

Pour plus de détails sur la façon de spécifier les cartes à imprimer, voir la commande **print** dans le dictionnaire de LiveCode.

Pour imprimer sur une échelle comprise entre 1 et 100% réglez **PrintScale** sur un nombre compris entre 0 et 1. Pour imprimer à 200% fixez **PrintScale** à 2.

Important : PrintScale s'applique à chaque carte que vous imprimez. Ce n'est pas lié à la **printPaperScale** qui est appliqué à l'ensemble du travail d'impression après que tous les autres calculs d'échelle aient été appliqués. Ainsi, vous pouvez régler la **printPaperScale** à 0,5 pour imprimer à 50%, puis imprimer des cartes individuelles à différentes valeurs **PrintScale**. Avec un **printPaperScale** de 0,5, et un **PrintScale** sur 2 fait que la carte imprime à 100%.

Lorsque vous imprimez une carte, utilisez **printMargins** pour spécifier les marges autour du bord de la carte dans la page.

Remarque : Lors du calcul de placement sur la page imprimée, tous les calculs supposent qu'il ya 72 points par pouce - indépendamment de la plate-forme ou du dispositif d'impression. LiveCode ajustera automatiquement l'imprimé pour la résolution du périphérique réel. Cela rend plus simple le calcul de votre mise en page.

```
set the printMargins is set to 72,72,72,72          -- Une marge d'un pouce de chaque côté
```

Important : printMargins s'applique uniquement lorsque vous imprimez la carte directement. Il n'a pas d'effet sur l'impression des cartes dans une mise en page (décrite ci-dessous).

La propriété **printCardBorders** spécifie si oui ou non la bordure en biseau sur le bord d'une carte devrait apparaître sur l'impression.

8.8.1 Options de Disposition de la Carte

Lorsque vous utilisez l'impression de base du format carte avec la commande d'impression, il existe deux options de mise en page qui vous permettent de personnaliser le positionnement des cartes sur la page imprimée. Si vous avez besoin d'une plus grande flexibilité, voir la section sur l'impression d'une mise en page, ci-dessous.

Utiliser la propriété **printRowsFirst** pour spécifier si les cartes doivent être imprimées en ligne puis vers le bas ou vers le bas puis en ligne.

Prenons un exemple simple d'impression de plusieurs cartes (cet exemple est utile pour l'impression d'étiquettes).
Dans cet exemple, nous avons une pile qui contient 8 cartes, chacune contenant une étiquette d'envoi. Si vous voulez essayer cet exemple :

- Créer une pile et la dimensionner à la taille d'une étiquette.
- Créer un unique champ, et dans l'inpecteur du champ, désactiver la propiété ***Shared Text***
- Grouper le champ et dans l'inspecteur de propriété du groupe activer ***Behave as*** **Background**
- Activez ***Select Grouped*** dans la barre de menu et sélectionnez le champ
- Placez le contenu de la première étiquette dans l'onglet ***Contents*** de l'inspecteur
- Créez 8 cartes supplémentaires, et dans chacune, sélectionnez le champ et placez le contenu des différentes étiquettes

Ainsi, nous avons une pile qui ressemble à figure ci-dessous :

Maintenant, nous allons mettre en œuvre les commandes d'impression. S'il s'agissait d'une véritable application, vous auriez probablement voulu placer une commande d'impression dans le menu Fichier. Dans ce cas, vous pouvez exécutez la commande suivante dans la boîte de message multi-lignes (Ouvrir la boîte de message, puis appuyez sur la deuxième icône pour obtenir le volet multi-ligne).

```
answer printer        -- permettre à l'utilisateur de choisir les options de sortie imprimante
print 9 cards
```

Appuyez sur Entrée pour exécuter les commandes.

La sortie d'impression résultante ressemblera à la figure ci-contre :

Si nous modifions commandes d'impression pour inclure une ligne supplémentaire pour désactiver **printRowsFirst** :

```
answer printer
set the printRowsFirst to false
print 9 cards
```

La sortie d'impression résultante ressemblera à la figure ci-contre :

Utilisez la propriété **printGutters** pour spécifier la marge entre chaque carte. Par défaut, **printGutters** est fixé sur 36,36 ou un demi-pouce horizontal et vertical.

Dans l'exemple suivant, nous imprimons la même pile d'étiquettes, mais en réduisant l'espace entre chaque étiquette au 1/10e de pouce. Pour faciliter l'aperçu des différences, nous activons également l'impression des bordures des cartes, en utilisant la propriété **printCardBorders**.

```
answer printer
set the printGutters to 7,7
set the printCardBorders to true
print 9 cards
```

La sortie d'impression résultante ressemblera à la figure ci-contre:

8.9 Impression de champs et textes

Pour imprimer un champ, utilisez la commande **revPrintField**. Cette commande ne prend qu'un seul argument, la référence à un champ. Cette commande permet de n'imprimer qu'un seul champ. Si vous avez besoin d'inclure un en-tête et pied de page ou le texte que vous avez construit par programme, consultez la commande **revPrintText** ci-après.

```
revPrintField the long id of field "text document"
```

Astuce : revPrintField est implémenté comme une bibliothèque de scripts situé dans l'IDE de LiveCode. La bibliothèque de script crée une pile invisible, définit le rectangle de cette pile au format du papier en cours, définit le **formatForPrinting**, sur vrai, crée un champ, puis copie le contenu du champ que vous indiquez dans cette pile invisible. Elle imprime ensuite ce champ, une page à la fois, en faisant défiler le texte après chaque page. Les utilisateurs avancés peuvent trouver ce script de la bibliothèque en allant à l'onglet ***Back Scripts*** dans ***Message Box***, en activant la case à cocher ***Show Live Code UI Back Scripts***, puis en éditant le script du bouton "**revPrintBack**". Le gestionnaire **revPrintField** est proche du sommet du script.

Utilisez la commande **revShowPrintDialog** pour contrôler si le système d'impression et les boîtes de dialogue de mises en page doivent être affichés par **revPrintField** ou **revPrintText**.

```
revShowPrintDialog true, false -- montrer la boîte de dialogue de l'imprimante système, mais pas la mise en page
revPrintField the long id of field "text document"
```

Utilisez la commande **revPrintText** pour imprimer du texte brut ou de style avec une entête et pied en option.

Pour plus de détails sur la façon de spécifier les cartes à imprimer, voir la commande **revPrintText** dans le dictionnaire de LiveCode.

8.10 Imprimer Une Mise en Page

Si vous avez besoin d'imprimer une mise en page plus complexe que ne l'autorise sur la commande de base d'impression de carte ou de texte (décrite ci-dessus), vous pouvez utiliser la syntaxe **print card into rect** pour créer tout type de mise en page selon votre choix.

```
print card from topLeft to rightBottom into pageRect
```

- **topLeft** est la coordonnée en haut à gauche de la carte en cours pour lancer l'impression depuis ce point.
- **rightBottom** est la coordonnée en bas à droite de de la carte actuelle pour stopper l'impression à ce point.
- **pageRect** est la surface rectangulaire sur du papier à imprimer.

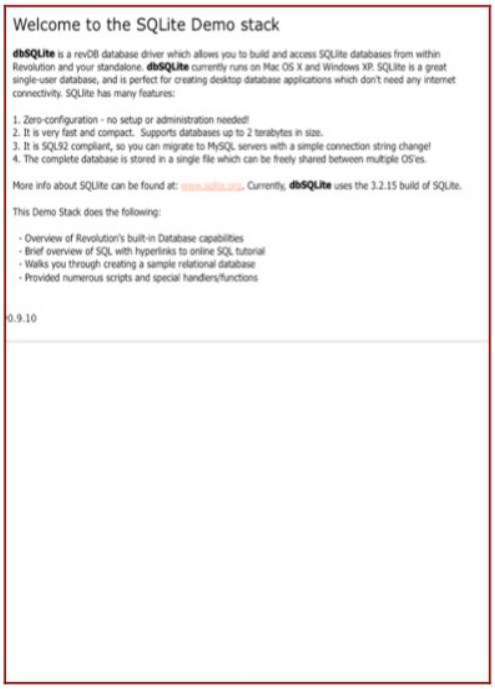
Important : printMargins s'applique uniquement lorsque vous utilisez **print card** directement. Il n'a pas d'effet sur l'impression des cartes dans une mise en page. Utilisez **printRectangle** pour obtenir la zone imprimable lorsque vous travaillez avec l'impression de mise en page.

Par exemple, nous voulons l'impression de la zone de texte à partir du milieu de la pile dans la figure ci-dessous. Vous pouvez charger la pile indiquée dans l'image en allant dans votre dossier d'installation de LiveCode puis en ouvrant **Tools > Ressources > Examples > SQLite Sampler.rev**
Nous voulons que la la sortie à l'échelle pour prendre toute la largeur du papier et la moitié de la hauteur.

Pile avec un champ de texte à imprimer dans une mise en page

```
print this card from the topleft of field "theText" to the bottomRight of field "theText"
into 0,0,item 3 of the printRectangle, round(item 4 of the printRectangle/2)
```

Cela se traduit par l'imprimé ci-dessous.



Résultat de la commande d'impression de mise en page

Vous pouvez construire une mise en page complexe en prenant des composants sur plusieurs piles par l'impression d'une suite de rectangles sur la même page. Par exemple, vous pouvez avoir une pile qui contient un en-tête standard et pied de page, une autre qui contient un logo et une mise en page qui contient du texte.

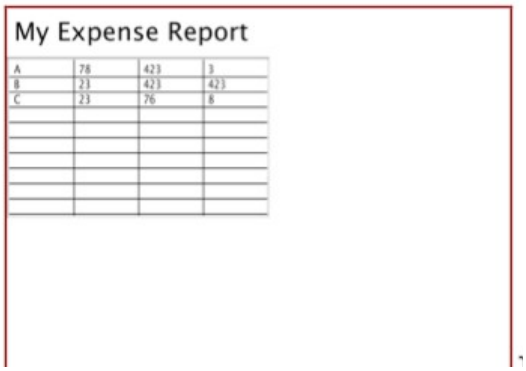
Utilisez la commande **open printing** pour commencer le travail d'impression, puis imprimer chaque élément dans rectangle approprié sur le papier. Ensuite, l'utilisation de la commande **close printing** pour envoyer le travail d'impression à l'imprimante.

L'exemple de la figure ci-dessous montre deux piles avec des zones imprimables que nous voulons combiner sur une seule feuille de papier.

Entête séparée du corps de la pile à imprimer dans une mise en page

Pour imprimer ceci sur une seule feuille :

```
answer printer -- boite de dialogue param impression du système
open printing   -- démarrer un tâche d'impression
set the defaultStack to "header" -- travailler avec la pile entête
print this card from the topLeft of field "header" to the bottomRight of field "header" into the rect of field "header" -- imprimer l'entête sur le coin haut gauche du papier
put the bottom of field "header" into tHeaderBottom -- sauver le résultat
set the defaultStack to "report editor" -- travailler avec la pile report editor
print this card from the topleft of field "report table" to the bottomRight of field "report table" into 0,tHeaderBottom,the right of field "report table", the bottom of field "report table" + tHeaderBottom -- imprimer le champtable en dessous de l'entête
close printing -- send the job to the printer
```



Résultat de la commande d'impression de mise en page avec plusieurs piles

8.10.1 Impression d'une mise en page complexe

Pour imprimer une mise en page plus complexe, créer une pile et définir son rectangle à la valeur **printRectangle** en cours.

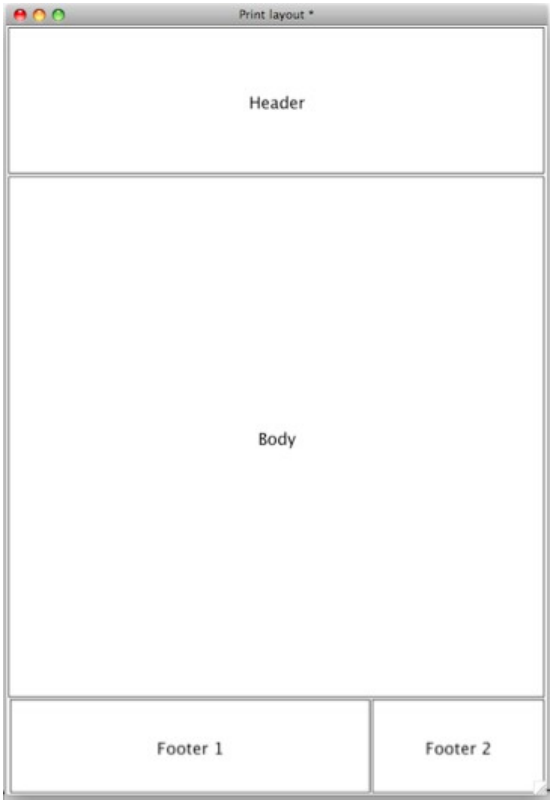
Ajouter des zones rectangulaires pour chaque composant que vous allez imprimer.

Puis définir les propriétés **Geometry** (voir la section sur le gestionnaire de géométrie pour plus d'informations) sur chacun de ces rectangles afin qu'ils se redimensionnent correctement lorsque la pile est retaillée.

Configurez votre programme d'impression pour ouvrir cette pile invisible puis la redimensionner à la valeur **printRectangle**. Cela va déclencher les routines de géométrie et la mise à l'échelle correcte des zones

rectangulaires. Ensuite, exécutez la séquence des commandes d'impression pour imprimer chaque rectangle.

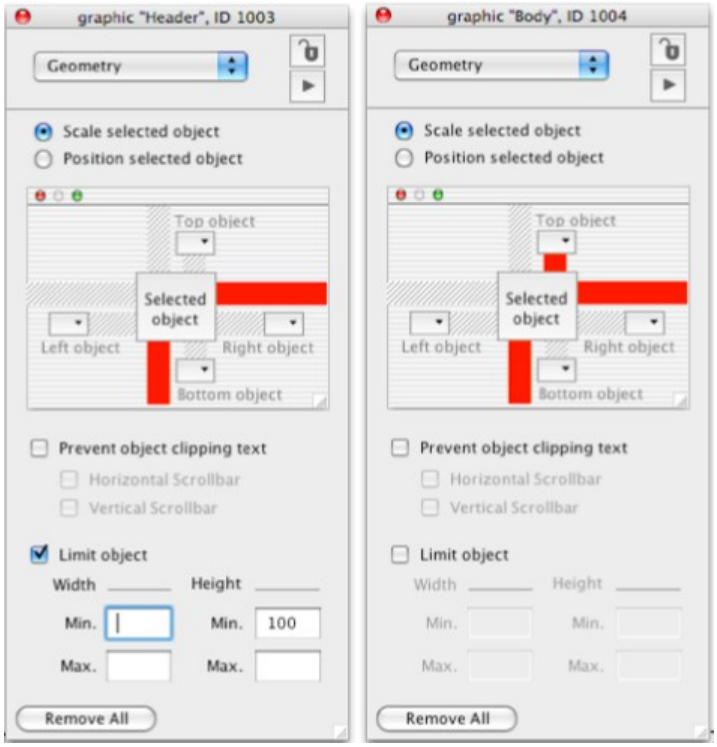
Dans la figure ci-contre, nous avons établi la taille de la pile à la valeur **printRectangle** et ensuite ajouté 4 graphiques rectangulaires. Nous avons nommé chaque graphique et activé la la propriété **Show Name** pour chacun, de sorte que vous puissiez voir leur nom.



Modèle de pile pour l'impression d'une mise en page

Ensuite, fixez les propriétés de géométrie pour chacun des graphiques rectangulaires.

Le graphique d'en-tête est réglé à l'échelle en fonction du bord droit puis inférieur, avec une limite de taille minimum de 100 pixels (voir figure ci-dessous).



Le graphique du corps est mis en lien avec le bord supérieur de et le bord l'entête graphique, et le bord droit puis inférieur de la pile (voir figure ci-contre).

Le pied de page 1 est réglé à l'échelle du bord droit positionné sur le bord inférieur.
Et pied de page 2 est positionné à la fois à droite et en bas.

Exemple : les propriétés Geometry pour la pile de mise en page

Pour créer le modèle d'impression de la pile à la taille du papier, ajouter le script suivant dans la pile :

```
on preOpenStack
  set the width of this stack to (item 3 of the printRectangle - item 1 of the printRectangle)
  set the height of this stack to (item 4 of the printRectangle - item 2 of the printRectangle)
end preOpenStack
```

Nous avons maintenant un modèle d'impression.

Tout ce qu'il reste à faire est d'écrire le script qui imprime dans les rectangles

```
-- Se préparer à charger cette pile hors de l'écran
hide stack "print layout"  -- ceci force le redimensionnement de la pile qui lance les routine de géométrie pour donner à
                                                                                                                chaque rectangle la bonne valeur
go stack "print layout"
                                -- maintenant stockons les cordonnées des rectangles dans des variables
put the rect of graphic "header" into tHeaderRect
put the rect of graphic "body" into tBodyRect
put the rect of graphic "footer 1" into tFooter1Rect
put the rect of graphic "footer 2" into tFooter2Rect
                                -- nous pouvons fermer la pile mise en page comme elle n'est plus nécessaire
close stack "print layout"
                                -- lancer la boîte de dialogue de l'imprimante système pour permettre à l'utilisateur de choisir le
                                                                                                                nombre de copies, etc
answer printer
                                -- démarrer la tâche d'impression
open printing
                                -- mettre la pile en cours d'utilisation dans une pile contenant l'en-tête
set the defaultStack to stack "header graphics"
                                -- imprimer à partir du rectangle de notre groupe entête, dans le rectangle que nous avons
                                                                                                                enregistré plus tôt
print this card from the topLeft of group "header" to the bottomRight of group "header" into tHeaderRect
set the defaultStack to "body contents"
print this card from the topLeft of group "body" to the bottomRight of group "body" into tBodyRect
set the defaultStack to "footer1"
print this card from the topLeft of group "footer1" to the bottomRight of group "footer1" into tFooter1Rect
set the defaultStack to "footer2"
print this card from the topLeft of group "footer2" to the bottomRight of group "footer2"
                                -- Envoyer le travail d'impression à l'imprimante
close printing
                                -- vérifier si l'utilisateur a annulé ou il y avait une erreur
if the result is “cancel” then
                               -- code à insérer ici pour traiter la condition d'annulation
else if the result is not empty then
                               -- faire apparaître une boîte de dialogue d'erreur
  answer “Printer Error”
else
                               -- insérer un code voulu ici pour gérer le succès
end if
```

8.11 Impression de plusieurs pages

8.11.1 pages multiples, en utilisant l'impression de cartes

Pour imprimer plusieurs pages lors de l'impression de cartes, spécifiez simplement les cartes que vous souhaitez imprimer, dans le cadre de la commande d'impression.

print {range}

Exemples:

```
print this card        -- imprimer la carte active
print all cards        -- imprimer toutes les cartes de la pile active
print 10 cards         -- imprimer les 10 cartes suivantes, en partant de la carte courante
print card 3 to 7      -- imprimer la carte 3 à 7 de la carte courante
print marked cards     -- imprimer toutes les cartes dont la propriété mark est sur vrai
```

8.11.2 Plusieurs Pages avec la Mise en Page d'impression

Pour imprimer plusieurs pages avec la mise en page d'impression, utiliser la commande **open printing** pour ouvrir une tâche d'impression. Ensuite, imprimez la mise en page pour la première page (voir ci-dessus). Ensuite, utilisez la commande **print break** pour insérer un saut de page dans le travail d'impression. Ensuite, insérez la deuxième page et ainsi de suite. Enfin, utilisez la commande **close printing** pour envoyer le travail d'impression à l'imprimante.

8.11.3 Imprimer une mise en page avec des champs à défilement

Pour imprimer un seul champ texte défilant, utilisez la commande **revPrintText** (voir ci-dessus pour plus d'informations). Si vous souhaitez intégrer le contenu d'un champ à défilement dans une mise en page, utilisez la propriété **pageHeights** pour faire défiler le champ à chaque fois que vous imprimez une page, puis la commande **print break** pour passer à la page suivante.
pageHeights renvoie une liste de valeurs indiquant dans quelle mesure un champ déroulant doit être parcouru pour empêcher le découpage d'une ligne de texte à chaque page de votre impression. (Vous devriez utiliser cette fonction conjointement avec la propriété **formatForPrinting** ci-dessus.)

```
put the pageHeights of field "body text" into tHeightsList     -- Mémoriser la liste
set the scroll of field “body text” to 0                       -- dérouler le champ depuis le début
open printing                                                  -- start the print job démarrer la tâche d'impression
repeat for each line x in tHeightsList          -- couper le champ à la partie inférieure de la dernière ligne visible
  set the height of field “body text” to x
  print this card from the topLeft of field "body text" to the bottomRight of field "body text" -- imprimer la zone texte
  print break
end repeat
close printing                                                 -- envoyer le travail à l'imprimante
```

Important : Définissez la propriété **LockLocation** (**lockLoc**) du champ sur vrai avant de régler la hauteur de la boucle ci-dessus, pour éviter que le champ "dérive" chaque fois que vous modifiez la hauteur.

Astuce : Désactivez les propriétés de la barre de défilement (**HScrollBar** et **VScrollBar**) des champs avant l'impression et régler la largeur de la bordure à 0 si vous voulez éviter l'impression d'une bordure ou d'une barre de défilement.

Astuce : Vous pouvez intégrer des champs déroulants, dans un modèle d'impression d'une pile de mise (voir la section Impression d'une mise en page complexe ci-dessus) pour rendre plus facile la gestion de l'impression d'une mise en page complexe. Créez un champ, dans votre modèle d'impression de la pile, désactivez la barre de défilement, réglez la largeur de la bordure à 0, l'emplacement du verrou sur vrai, puis les propriétés géométriques comme pour la section ci-dessus. Au début de chaque travail d'impression, copier la police du texte et sa taille en utilisant les propriétés **textSize** et **textFont** ensuite le contenu du champ de texte que vous souhaitez imprimer à l'aide de la propriété **htmlText**.

8.11.4 Travailler avec des Zones d'impression

Utilisez la propriété **printRanges** pour obtenir une liste des pages que l'utilisateur a sélectionné dans la boîte de dialogue des réglages de l'imprimante. Utilisez cette propriété lors de l'impression pour éviter l'impression de pages que l'utilisateur n'a pas sélectionné.

Pour utiliser cette propriété, ouvrir une boîte de dialogue d'impression du système, puis stocker les **printRanges** dans une variable. Ensuite, réglez le **printRanges** à "**all**", puis envoyez uniquement les pages qui ont été sélectionnés (stockées dans la variable) à l'imprimante.

Remarque : Si vous ignorez la propriété **printRanges**, LiveCode va gérer ce paramètre automatiquement. Il suffit d'envoyer chaque page à l'imprimante comme normalement et LiveCode ignorera les pages que l'utilisateur n'a pas sélectionné dans la boîte de dialogue d'impression. Ne gérez manuellement cette option que si vous imprimez une mise en page extrêmement complexe et que vous voulez gagner le temps de traitement pour l'élaboration de la mise en page de chaque page non sélectionnée.

Utilisez le **printPageNumber** pour obtenir le numéro de la page en cours d'impression au cours de votre boucle d'impression.

8.12 Imprimer Un Objet du Navigateur

Pour imprimer le contenu d'un objet du navigateur, utilisez la commande **revBrowserPrint**. Pour plus d'informations, voir la commande **revBrowserPrint** dans le Dictionnaire du LiveCode.

Chapitre 9 Déploiement de votre application

Avec la capacité de LiveCode de générer des programmes autonomes vous pouvez créer une application bureautique native pour chaque système d'exploitation que vous souhaitez prendre en charge. Les utilisateurs qui n'ont pas LiveCode peuvent exécuter ces applications comme toute autre application qu'ils téléchargent et installent. Les applications autonomes peuvent avoir leur propre identité comme de véritables applications, inclure une icône sur le bureau, des associations de documents et plus encore.

9.1 Création d'une application autonome

Lorsque vous avez terminé votre application LiveCode et pour pouvoir la distribuer, vous pouvez l'intégrer dans une application autonome. Ces applications ne nécessitent pas que les utilisateurs possèdent LiveCode.

Tout l'ensemble des fonctionnalités de LiveCode, est disponible pour une utilisation dans une application autonome, à l'exception de ce que vous ne pouvez pas définir par des scripts dans les objets.

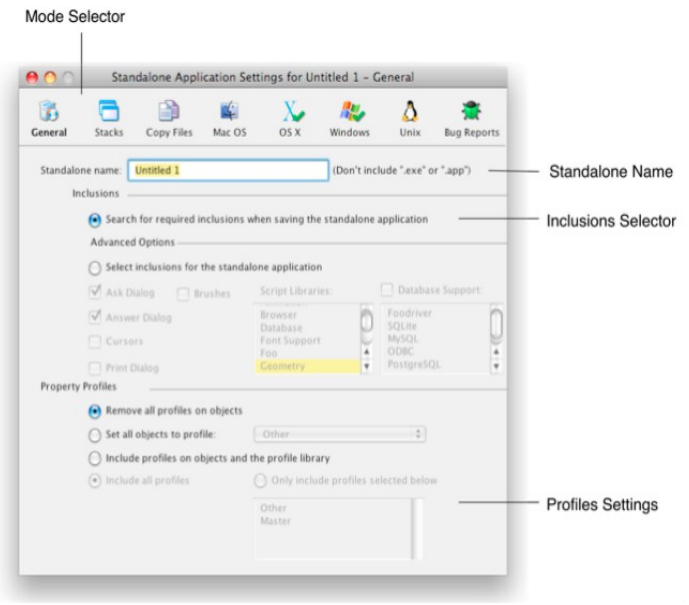
L'outil de construction lui-même va vous permettre de créer des applications autonomes pour toute plate-forme prise en charge (par exemple, vous pouvez construire une application Windows autonome sur une machine Mac OS X).

Toutefois, vous pouvez souhaiter vérifier que votre application s'accorde et se comporte correctement sur chaque plate-forme que vous souhaitez soutenir. Veuillez noter qu'il est intrinsèquement plus difficile à déboguer une application qui a été construite comme une application autonome, donc vous devriez tester votre application aussi complètement que possible avant de la compiler.

9.1.1 Paramètres des applications autonomes

La boîte de dialogue de configuration des applications autonomes permet de créer des réglages pour votre application autonome.
Cette boite de dialogue peut être trouvée dans le menu Fichier.
Les paramètres que vous saisissez sont appliqués à la pile actuelle la plus éditable et au dessus, et sont enregistrés avec.
Cela signifie que vous ne devez saisir les paramètres qu'une fois pour chaque application que vous créez.
Les mêmes paramètres s'appliqueront si vous ne configurez pas une autre construction à l'avenir.

Paramètres Constructeur des *Standalone* - Onglet Général



Mode Selector Choix entre différents écrans de réglages des applications autonomes

Standalone Name Définissez le nom de votre application autonome. Cela devrait être le nom voulu pour toute application finalisée. Ne pas inclure une extension de fichier (.exe sur Windows ou .app sur Mac OS X) puisque le constructeur peut créer des logiciels autonomes (**standalones**) pour de multiples plates-formes. Il ajoutera lui-même l'extension appropriée.

Inclusions Selector Choisissez les éléments que vous souhaitez inclure dans votre application autonome. Vous pouvez soit choisir de rechercher les inclusions nécessaires automatiquement, ou sélectionner manuellement les composants que vous souhaitez inclure.

Search for Inclusions C'est l'option par défaut. Lorsque sélectionnée, LiveCode va chercher votre fichier de pile de l'application (pile principale et sous des piles) pour tenter de déterminer les composants utilisés par votre application. Il inclura alors ces éléments.

Select Inclusions for the Standalone Applications Sélectionnez cette option si vous souhaitez spécifier les éléments à inclure manuellement. Vous pouvez utiliser cette option si votre application charge dynamiquement les composants qui ne peuvent être déterminés à ce point automatiquement, ou si vous savez exactement quels composants utilise votre application et que vous souhaitez accélérer le processus de construction de l'application autonome en sautant l'étape de recherche automatique.

Il est important que vous choisissiez d'inclure tous les composants utilisés par votre application ou bien cette opération peut échouer. Si vous ne n'incluez pas votre propre rapport d'erreur personnalisé ou boîte de dialogue de rapport d'erreurs pour votre application autonome LiveCode (voir ci-dessous) un échec peut rester silencieux - à savoir une opération dans votre application autonome va tout simplement cesser de travailler sans rien indiquer à l'utilisateur.

Ask Dialog Cette option est nécessaire si un de vos scripts utilise **ask** ou la commande **ask password**. Le constructeur pour application autonome va copier la pile **ask dialog** depuis l'IDE dans votre **standalone**, comme une sous pile. Le constructeur autonome effectue une copie de votre pile avant l'ajout de ressources à elle dans le cadre du processus de construction, de sorte que votre pile d'origine n'est pas modifiée.

Answer Dialog Cette option est nécessaire si un de vos scripts utilisez la commande **answer**. Notez que cela ne s'applique qu'à la forme de dialogue de la commande. **Answer file / printer / color / effect / folder / page setup / printer** et les formes record de la commande ne nécessitent pas cette option. Le constructeur standalone va copier la pile **answer dialog** dans votre application autonome.

Cursors Cette option est requise si votre application utilise un des curseurs de LiveCode. Ce n'est pas nécessaire si votre application utilise uniquement des curseurs OS. Il copie les curseurs de la pile dans votre autonome.

Print Dialog Cette option est requise si votre application utilise la boite de dialogue d'impression intégrée ou une page de réglages (par exemple pour une utilisation sur Linux sans GTK installé). Ce n'est pas nécessaire si vous affichez uniquement l'imprimante système et la page des dialogues de configuration. Il copie la pile "**print dialog**" et "**page setup**" dans votre application autonome.

Brushes Cette option est requise si votre application utilise un des curseurs de pinceau de LiveCode. Ce n'est pas nécessaire si votre application ne fait pas usage des commandes de peinture. Il copie la pile

brushes dans votre application autonome.

Script Libraries Cette option vous permet de copier des bibliothèques de scripts dans votre application. La liste des bibliothèques disponibles est automatiquement mise à jour afin d'inclure tout plug-in de bibliothèques ou externes que vous avez installé dans la distribution de LiveCode. Ainsi, la liste que vous avez peut différer de ce qui est documenté ici.

Lorsqu'elle est incluse dans une application autonome, chaque bibliothèque de scripts est implémentée comme un groupe caché et mise à disposition lorsque le groupe reçoit son premier message de **openBackground**. Pendant la première partie du processus de démarrage de l'application avant que ce message ne soit envoyé, toutes les commandes qui utilisent une bibliothèque donnée ne seront pas disponibles. Cela peut affecter les tentatives d'utiliser cette bibliothèque de script de démarrage, **preOpenStack**, **OpenStack**, ou les gestionnaires **preOpenCard** dans la pile principale. Une fois que l'application a terminé la mise en service, la bibliothèque est disponible et la bibliothèque de script peut être utilisée dans n'importe quel gestionnaire.

Animation Cette bibliothèque est non pris en charge.

Browser Navigateur intégré et une commande de **revBrowser**.

Database Accès aux bases de données et une commande de **revDatabase**.

Font Support revFontLoad et **revFontUnload**.

Geometry Propriétés de géométrie ou commandes.

Internet Printing L'accès à Internet, y compris les **URL**, **ftp** et **POST**, **revPrintField**, **revShowPrintDialog** et **revPrintText**

LiveCode Zip Toutes les commandes **revZip** (mais non nécessaire pour compresser / décompresser)

Speech revSpeak et **revSpeechVoices**

SSL & Encryption Toutes les commandes de cryptage **SSL** ou connexes

table Utilisation de l'objet de table

Video Grabber Toutes les commandes de capture vidéo

XML Toutes les commandes **revXML**

XMLRPC Toutes les commandes **revXMLRPC**

Database Support Cette option est nécessaire si vous utilisez des bases de données **SQL**. Assurez-vous que vous choisissez d'inclure les pilotes pour toute autre base de données à laquelle vous avez besoin d'accéder.

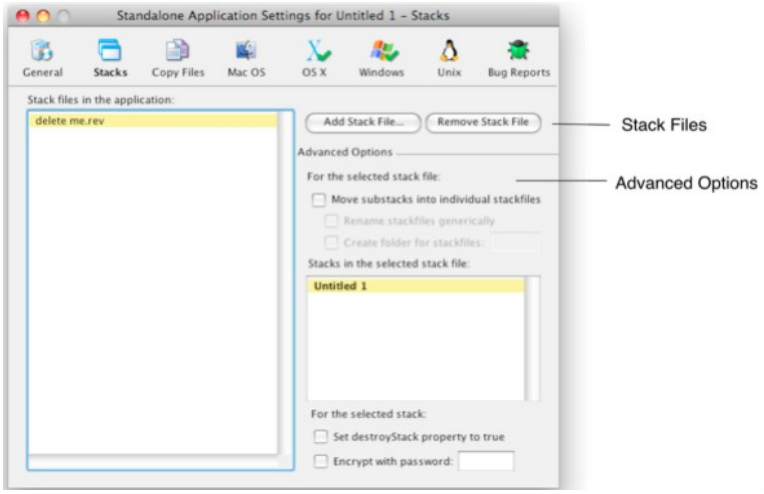
Profiles Settings Choisissez entre les paramètres des options **Property Profile**. Il vous suffit de modifier les paramètres dans cette zone si vous avez utilisé **Property Profiles** (voir la section sur les profils de propriétés dans le chapitre 4, le constructeur d'une interface utilisateur ci-dessus)

Remove all profiles Supprime tous les profils et construit les autonome à l'aide du profil actif sur chaque objet. Sélectionnez cette option si vous n'avez pas besoin de changer de profil dans la version autonome et que vous souhaitez économiser de l'espace disque en supprimant les informations de profil étrangères.

Set all objects to profile Réglez tous les objets à un profil spécifique puis retirez les données du profil des objets.

Include profiles and the profile library Inclure les bibliothèques de profils et permettre la commutation entre les profils dans l'application autonome. Vous pouvez choisir d'inclure des profils spécifiques ou tous les profils.

Onglet Piles
Configurations Application Autonome



Stack Files Utilisez cette section pour ajouter les fichiers de piles supplémentaire à votre application. Toutes les piles que vous ajoutez à cette section seront ajoutées à la propriété **stackFiles** de la pile principale de votre application. Cela signifie que tous les scripts dans votre application autonome pourront localiser et référencer ces piles par leur nom.

Advanced Options Utilisez cette section pour contrôler exactement comment de multiples fichiers de piles sont gérés dans votre application.

Move substacks into individual Si vous sélectionnez cette option, chacune des sous piles dans les fichiers de la pile que vous sélectionnez seront déplacés dans leur propre dossier individuel, situé dans le dossier de données ou dans le paquet application de votre programme.

Rename stackfiles generically Renomme chaque fichier de sous-pile à l'aide d'un numéro sur le disque (au lieu d'utiliser le nom de la sous-pile). Sélectionnez cette option si vous ne voulez pas que les noms que vous avez sélectionnés pour les piles soient visibles à l'utilisateur final dans le système de fichiers.

Create folder for stackfiles Crée un dossier et place les fichiers de la pile dans ce dossier, au lieu de les stocker au même niveau que le fichier exécutable du programme. Toutes les références dans la propriété stackFiles se référeront à ce dossier en utilisant un chemin relatif afin que les piles puissent encore être localisées par l'application autonome.

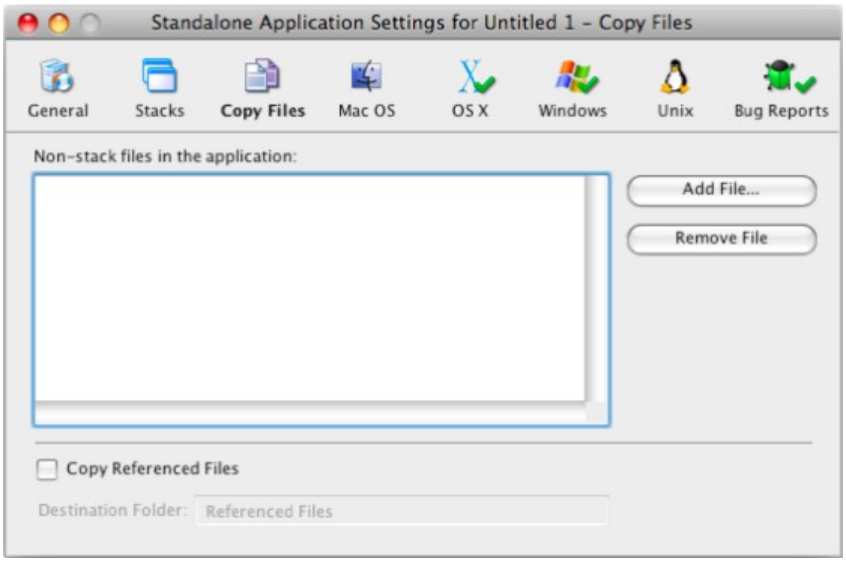
Individual stack options Sélectionnez un fichier de pile sur la gauche puis une pile individuelle depuis ce fichier pour définir les options sur cette pile.

Set destroyStack to true Paramétrez cette option si vous voulez que le fichier de pile sélectionné soit supprimé de la mémoire quand il est fermé. Cette option est utile si vous chargez de grandes piles en mémoire et que vous souhaitez qu'elles soient retirées lorsqu'elles sont fermées.

Encrypt with password Sécurise le script dans le fichier de pile sélectionné avec un mot de passe. Cela fournit un niveau de base de cryptage qui empêche quelqu'un de lire avec désinvolture les scripts dans la pile en ouvrant le fichier dans un visualiseur de fichier binaire.

Remarque : Un fichier de pile directement attaché à une application autonome ne peut pas subir des changements sauvegardés en son sein. Cette pile est liée directement au fichier exécutable qui fonctionne. Le système d'exploitation verrouille un fichier exécutable en cours d'exécution. Si vous souhaitez enregistrer les modifications dans votre application autonome, fractionnez votre pile en plusieurs fichiers. Une technique courante consiste soit de créer une pile dite écran d'accueil qui contient un affichage de bienvenue, puis charge les piles qui composent le reste de votre application. Ces piles sont référencées comme appartenant à **stackFiles** sur ce plan dans la configuration du programme. Il est ainsi possible de mettre à jour automatiquement les composants des piles ou d'enregistrer des modifications. Vous pouvez également envisager de créer des fichiers de préférences dans l'emplacement approprié sur le système de votre utilisateur final (voir la fonction **specialFolderPath** et les fonctions **query/setRegistry** pour plus d'informations).

Paramètres pour *Standalone* Copier des fichiers



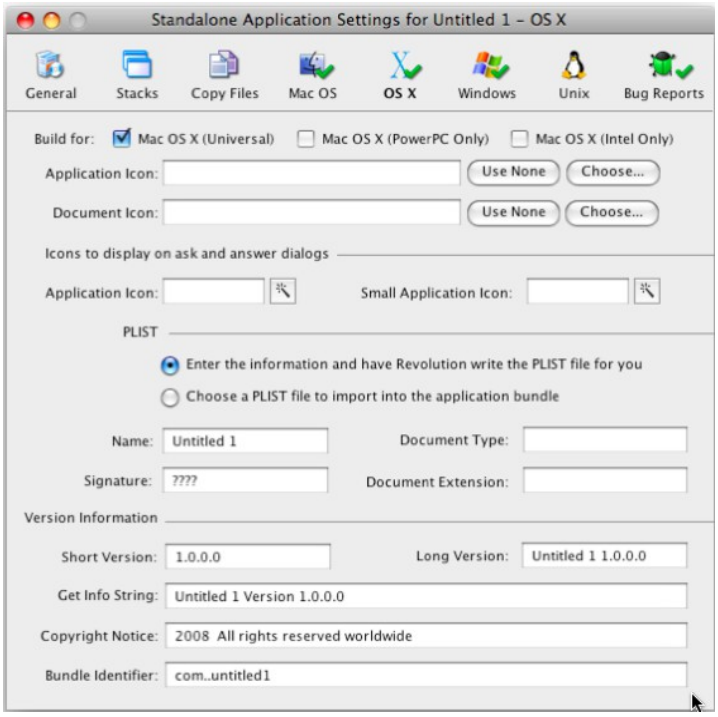
Non-stack files in the application Indiquer d'autres fichiers à inclure dans la version autonome. Utilisez cette option pour inclure les documents d'aide, le texte me lire, des fichiers et d'autres ressources que vous souhaitez inclure dans votre programme chaque fois que vous le générez.

Copy Referenced Files Liste tous les images et objets lecteurs dans les piles et copie tous les fichiers référencés dans la propriété **fileName** de ces objets dans la version autonome. Puis il règle automatiquement la propriété **fileName** pour faire référence à ces fichiers dans la version autonome en utilisant les chemins de fichiers référencés.

Destination folder Créez un sous-dossier dans votre programme pour y copier les fichier images et films.

Paramètres Application Autonome Mac OS Classic (non traité car trop ancien)

Paramètres Application Autonome OS X



Mac OS X (Universal) Construire un programme pour Mac OS X en format binaire universel. Ce programme sera exécuté nativement sur les processeurs Intel et PowerPC machines.

Mac OS X (PowerPC Only) Construire un programme qui va s'exécuter en mode natif sur Mac OS X PowerPC (Ancien)

Mac OS X (Intel Only) Construire un programme qui va s'exécuter en mode natif sur Mac OS X Intel. Ce programme ne fonctionne pas du tout sous PowerPC.

Application Icon Choisissez une icône d'application pour représenter l'application dans le Finder. L'icône doit être au format **icns**.

Document Icon Choisissez une icône de document pour représenter les documents de votre application dans le Finder. L'icône doit être au format **icns**.

Icons for ask / answer dialogs Choisissez une icône à afficher lorsque vous utilisez les commandes question ou réponse à afficher une boîte de dialogue. Sur Mac OS X, la convention veut que ces dialogues devraient afficher l'icône de votre application. L'icône doit être stockée dans votre pile comme une image, ou

sélectionnée à partir des icônes intégrées dans LiveCode. Si vous avez utilisé une icône intégrée, veillez à sélectionner l'inclusion pertinente sur l'onglet Général (si vous sélectionnez de les inclure manuellement).

PLIST – enter information and have LiveCode write the PLIST LiveCode complète les **PLIST** pour votre application automatiquement. Le **PLIST** est un fichier de paramètres stockés au format **XML** rangé au sein de chaque application Mac OS X. Il contient des informations sur l'application, y compris son nom, son numéro de version, copyright et les associations de documents. Permettre à LiveCode de créer ce fichier pour vous est l'option recommandée.

Choose a file to import into the application bundle Choisissez d'importer un fichier **PLIST** au lieu de laisser LiveCode en créer un. Sélectionnez cette option si vous avez créé votre propre **PLIST** hautement personnalisé que vous souhaitez utiliser pour votre application dans chaque **build** que vous créez.

Short version / long version Les informations de version à inclure avec votre programme.

Get info string Le texte visible qui s'affiche dans fenêtre Obtenir des informations de votre application depuis le Finder.

Copyright notice La notice copyright de votre application.

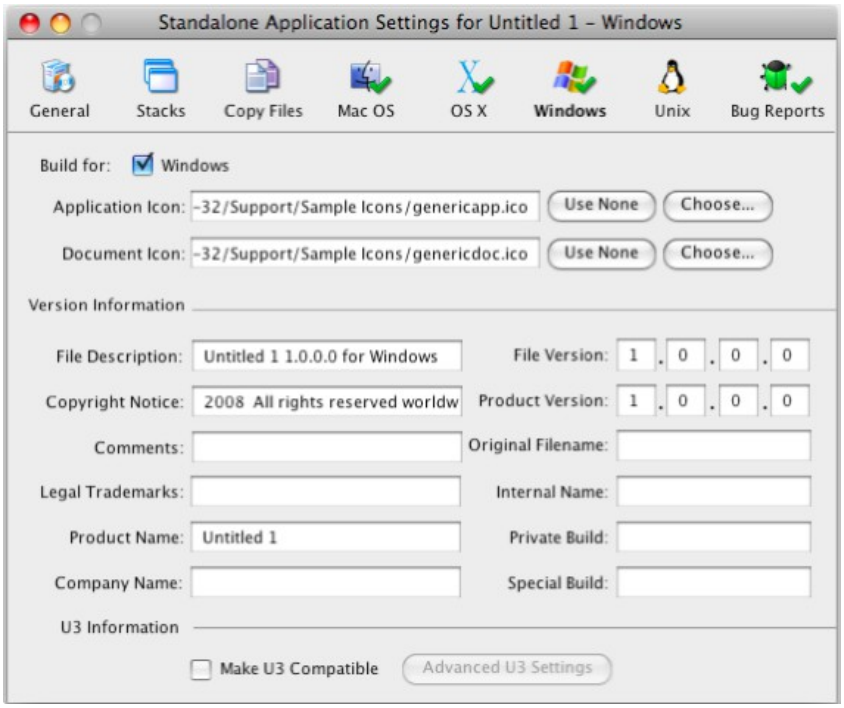
Bundle identifier Un identifiant unique pour votre application utilisé par Mac OS X pour identifier votre application.

Paramètres Application Autonome Windows

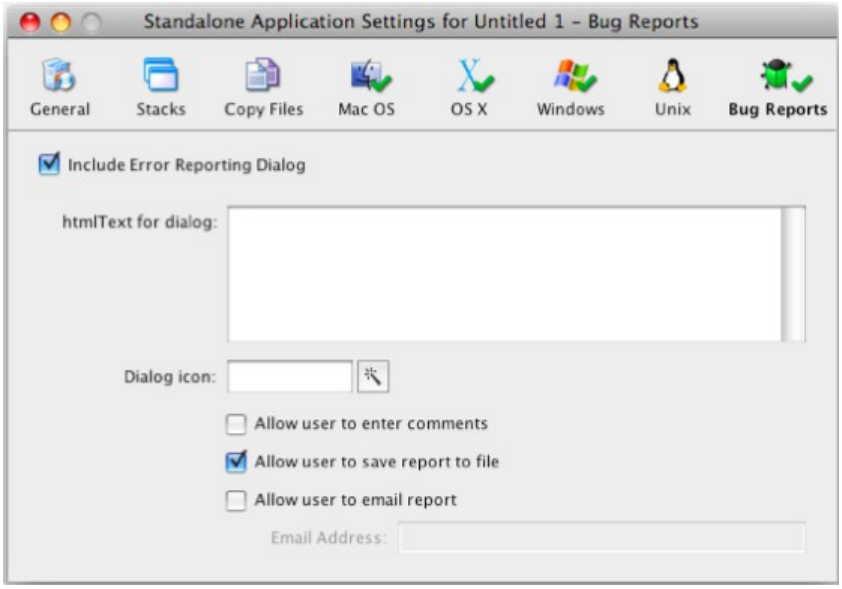
Build for Windows Application icon Construire un programme pour Microsoft Windows.

Document icon Choisissez une icône d'application pour représenter l'application sous Windows. L'icône doit être au format **.ico**.

Version information Choisissez une icône de document pour représenter les documents de votre application sous Windows. L'icône doit être au format **.ico**.



Paramètres Application Autonome Rapports de bogues



Include Error Reporting Dialog Inclure une pile de rapports d'erreurs dans votre programme. Vous devez sélectionner cette option si vous testez votre application et que vous souhaitez les détails des erreurs dans votre programme, ou si vous n'avez pas prévu vos propres routines de rapport d'erreurs dans vos piles.

htmlText for dialog Le texte à afficher à l'utilisateur dans la boîte de dialogue qui apparaît lorsqu'une erreur est rencontrée. Ce texte doit être en format HTML compatible LiveCode. Créer et formater le texte dans un champ de LiveCode puis copiez la propriété HTMLText du champ.

Dialog icon L'icône à afficher dans la boîte de dialogue d'erreur. Elle doit être conservée comme une image dans votre pile.

Allow user to enter comments Affiche une boîte pour que l'utilisateur vous donne plus d'informations. Cette information sera incluse dans le rapport.

Allow user to save report to file Permettre à l'utilisateur d'enregistrer le rapport dans un fichier. Sélectionnez cette option si vous souhaitez que les utilisateurs enregistrent un rapport d'erreur et vous l'envoient.

Allow user to email report Permettre à l'utilisateur d'envoyer le rapport. Sélectionnez cette option si vous voulez que l'utilisateur puisse envoyer à votre service de support technique, les détails de l'erreur. Cette option charge le client de messagerie par défaut du système et remplit l'e-mail avec le contenu du rapport d'erreur et les commentaires de l'utilisateur.

To : ce champ indique l'adresse email à joindre.

Chapitre 10 Gestion des erreurs et débogage

Dans un monde idéal, tout le monde saurait écrire le code parfait et il il ne serais jamais nécessaire de le déboguer. Cependant, en réalité, pratiquement tous les projets vont nécessiter un certain degré de débogage. Du plus complexe au projet le plus simple, cela reste vrai. Heureusement LiveCode comprend une multitude d'outils et de techniques qui font qu'il est rapide et facile à traquer les erreurs débogage. Le cycle de exécution/édition direct vous permet de voir l'effet des corrections dès que vous en apportez. Et, contrairement au travail dans un langage de bas niveau, quand vous faites une erreur, vous recevrez un message d'erreur compréhensible vous indiquant, l'endroit où l'erreur s'est produite, plutôt que l'application quitte inopinément.

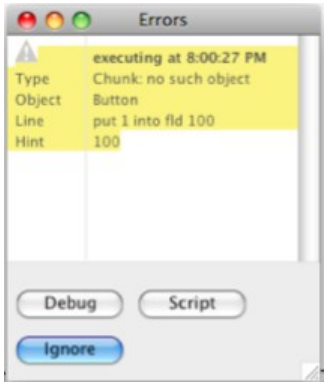
Ainsi avec la gestion d'erreur intégrée, le reporting et les outils de débogage, LiveCode vous permet également une flexibilité totale sur la gestion des erreurs, vous permettant de fournir une expérience personnalisée à l'utilisateur final de votre application.

10.1 Techniques Communes pour Résoudre les Problèmes

Si vous rencontrez un problème avec votre code, il y a un certain nombre de méthodes disponibles pour vous aider à le repérer. Dans cette section, nous détaillons quelques-unes des principales techniques qui peuvent vous être utiles.

10.1.1 La boîte de Dialogue d'erreur de LiveCode

Souvent, la première chose qui va vous alerter d'un problème est un message d'erreur.



Il existe deux types possibles de dialogue d'erreur. Le premier est le dialogue d'erreur d'exécution. Cette boîte de dialogue s'affiche lorsque votre script est exécuté et rencontre une erreur.

Dans l'exemple ci-dessus, le script tentait d'accéder au champ 100. Ce champ n'existe pas. Quand LiveCode rencontre un problème comme celui-ci, l'exécution s'arrête et le dialogue ci-dessus s'affiche.
Si vous savez quelle est l'erreur affichée, utilisez le bouton **Script** pour aller directement au script et le modifier. La ligne qui a généré l'erreur sera mise en évidence dans la fenêtre de script.
Alternativement, si vous avez besoin de plus amples informations, le bouton **Debug** va charger le script dans le débogueur sur la ligne de l'erreur (voir ci-dessous pour plus d'informations sur l'utilisation du débogueur). Vous pouvez ensuite charger l'observateur de variables (voir ci-dessous) pour voir l'état de toutes les variables à l'endroit où l'exécution a été interrompue.

Remarque : Le bouton de débogage n'apparaîtra que si **Script Debug Mode** est coché dans le menu Développement.

Erreurs lors de la compilation

Une boîte de dialogue d'erreur de script s'affiche lorsque votre script ne peut pas être compilé à cause d'une erreur de syntaxe. Cette boîte de dialogue s'affiche généralement lorsque vous tentez de compiler un changement dans un script en appuyant sur le bouton **Apply** dans l'éditeur de code. Appuyer sur le bouton **Script** permet de sélectionner la ligne qui a provoqué l'erreur. Corrigez l'erreur, puis appuyez sur le bouton **Apply** pour compiler à nouveau le script.

Attention : Si une erreur de compilation est générée alors l'ensemble du script auquel il s'applique ne sera pas compilé, et pas seulement la ligne ou le gestionnaire qui contient l'erreur. Si d'autres scripts tentent d'appeler des commandes ou fonctions dans celui-ci ils ne seront pas en mesure de s'exécuter jusqu'à ce que vous corrigiez le problème et compilier à nouveau le script.

Parce que LiveCode compile tous les scripts dans une pile quand il les charge, une boîte de dialogue d'erreur de script peut également être générée lorsque vous chargez une pile à partir d'un disque pour la première fois - si vous avez enregistré une pile qui contient un script qui ne peut pas se compiler.

Important : Ne confondez pas l'erreur d'exécution et les dialogues de d'erreur de script. La boîte de dialogue d'erreur d'exécution se produit lorsqu'un script est exécuté et ne peut pas continuer en raison d'une erreur. La boîte de dialogue d'erreur va commencer avec les mots "**executing at [time]**". La boîte de dialogue d'erreur de script s'affiche lorsque vous tentez de compiler un script qui contient une erreur de syntaxe. La boîte de dialogue d'erreur va commencer avec les mots "**compiling at [time]**". La boîte de dialogue d'erreur de script ne pourra jamais contenir un bouton de débogage parce que le script qui a généré l'erreur n'est pas en cours d'exécution.

Astuce : Si vous activez l'option vérification des variables pour l'éditeur de code, LiveCode exigera que vous déclariez toutes les variables avec toutes les chaînes littérales entre guillemets. Il signalera alors une erreur de compilation du script si vous essayez d'utiliser un littéral sans les **" "** ou de ne pas déclarer une variable. Cela peut être utile pour capturer les erreurs avant d'exécuter un script. Notez que si vous activez cette option

et essayez de compiler un script existant qui n'a pas été écrit de cette façon, il va typiquement générer un grand nombre d'erreurs qui à corriger avant de pouvoir compiler.

10.1.2 Supprimer Erreurs et Messages

Si votre pile entre dans un état instable car elle génère un grand nombre d'erreurs, vous pouvez désactiver temporairement l'envoi de messages à la pile ou l'affichage de messages d'erreur. Cela peut vous permettre de modifier la pile pour faire les changements appropriés.

Pour supprimer les messages, appuyez sur le bouton **Messages** sur la barre d'outils ou choisissez **Suppress Messages**, dans le menu **Development**. Les messages d'événements système cesseront d'être envoyés à la pile (par exemple un clic sur un bouton ne déclenche plus un gestionnaire **mouseUp**, changer de carte ne déclenche plus le gestionnaire **openCard**). Vous pouvez quand même envoyer des messages personnalisés directement aux objets à l'intérieur de la pile.

Pour supprimer les erreurs, appuyez sur le bouton **Errors** sur la barre d'outils ou choisissez **suppress Errors** dans le menu **Development**. Cela permettra d'éviter d'éventuelles erreurs à partir du déclenchement d'une fenêtre d'affichage d'erreur.

Attention : N'oubliez pas de réactiver les messages et les erreurs lorsque vous avez terminé l'édition. Sinon, votre pile ne fonctionnera pas, ou toute erreur qui apparaissant pendant le fonctionnement de la pile provoquera un arrêt mais n'affichera pas de message d'erreur.

10.1.3 Informations Pendant le Fonctionnement de Votre Script

Si vous voulez connaître l'état d'une variable ou un état particulier lors de l'exécution, vous pouvez utiliser l'observateur des variables, détaillé ci-dessous. Cependant, parfois, vous voudrez peut-être exécuter un script sans avoir à ouvrir le débogueur, mais avoir en sortie les informations au fur et à mesure.
Vous pouvez utiliser la commande **put** pour sortir les informations vers la **Message Box**, dans un champ ou un fichier texte, la commande **write** pour une sortie vers la console ou la commande **answer** pour afficher une boîte de dialogue.

Sortie vers la Message Box

MessageBox est un moyen pratique pour une sortie des informations tandis qu'un script est exécuté. Il vous permet d'afficher les informations que vous souhaitez sans avoir à créer un champ ou une pile pour afficher les informations. En utilisant une commande **put** sans spécifier de destination vos sorties sont dirigées vers **MessageBox** :

```
put tVar
```

Dans l'exemple ci-dessus, remplacer tVar avec le nom de la variable dans le script dont vous voulez voir le contenu. Vous pouvez également exporter des informations uniquement si une certaine condition est remplie :

```
if tVar is true then put tData
```

Chaque fois que vous affichez quelque chose dans la boîte de message, un message appelé variable globale spéciale (souvent abrégé en **msg**) est mis à jour. Cela vous permet de vérifier le contenu de la boîte de message facilement, ainsi que d'ajouter ou d'annexer une information plutôt que de la remplacer.

```
put tInformation & return after msg -- place tInformation et return après les données déjà dans la message box
```

Affichage sur la sortie standard (stdOut ou la console sur Mac OS X)

La sortie standard est un lieu utile pour enregistrer des messages. Contrairement à la boîte de message, il est facile d'identifier une séquence d'événements que vous pourrez ensuite retrouver ultérieurement. Elle a également l'avantage d'être extérieure à LiveCode, donc son utilisation n'interfère pas avec votre fenêtre superposée ou un focus quelconque. La sortie standard est disponible uniquement sur les systèmes Mac OS X, Linux ou Unix. Sur Mac OS X, elle peut être consultée en ouvrant l'application Console, située dans le dossier **Applications /Utilitaires**.
La syntaxe, pour écrire quelque chose sur la sortie standard ou la console est :

```
write tMessage & return to stdout
```

Astuce : Si vous écrivez beaucoup de données sur la console, il peut être utile d'ajouter l' heure à chacune pour les rendre plus facile à déboguer. Si vous avez besoin de plus granularité que les secondes, utilisez les millisecondes plutôt que la longue durée.

```
write tMessage && the long time & return to stdOut.
```

Astuce : Si vous insérez des instructions de sortie de débogage dans votre code, envisagez de les rendre conditionnelles à une variable globale. Cela vous permet d'activer le débogage en définissant la variable sans apporter de modifications au code. Encore mieux, il empêche d'oublier le code de débogage dans votre application, et par conséquent, le remplissage accidentel de la console du système de l'utilisateur final, avec des messages.

```
if gDebugging then write tMessage & return to stdOut
```

Sortie dans un champ

Vous pouvez créer une pile qui contient les champs utilisés pour le débogage :

```
put tVar & return after field "debugging info" of stack "my debug stack"
```

Astuce : Vous pouvez créer un ensemble d'outils qui facilite le débogage d'une application spécifique sur laquelle vous travaillez. Créer cette pile puis enregistrez-la dans votre dossier de plug-ins de sorte qu'elle soitt disponible dans le menu **Development** de l'IDE.

Sortie vers une boîte de dialogue

Pour afficher une boîte de dialogue avec le contenu d'une déclaration, utilisez la commande **answer**. Cette méthode est idéale si vous souhaitez afficher quelque chose rapidement, mais elle est généralement impropre à des tâches de débogage plus grandes puisqu'elle suspend l'exécution pour afficher une boîte de dialogue à chaque fois.

```
if tVar is true then answer tVar
```

Sortie dans un fichier texte

Si vous souhaitez enregistrer les informations de façon plus permanente, vous pouvez utiliser un fichier texte. Vous pouvez stocker le chemin du fichier dans une variable globale de sorte que vous pouvez le changer facilement. Voir la section sur l'utilisation des **URL** de fichiers ci-dessous pour plus d'informations.

```
put tVar & return after URL "file:debug.txt"
```

10.1.4 Interrompre Une Exécution

Si vous devez interrompre un script en cours d'exécution, appuyez sur **Ctrl-period** (.) (ou **command-period** (.) sous Mac OS). Notez que l'interruption d'un script ne fonctionnera que si la propriété globale **allowInterrupts** est réglée sur vrai.

Astuce : Sur les systèmes Mac OS X, si votre application est rentrée dans un état instable et que vous ne parvenez pas à l'interrompre avec **command-period**, vous serez en mesure de l'interrompre et de reprendre le contrôle en lui envoyant un signal. Ouvrez l'utilitaire Terminal, puis utilisez la commande **top-o cpu** ou **ps ax** pour récupérer l'**ID** de processus pour LiveCode. Ensuite, exécutez la commande **kill-sighup [processID]** où **[processID]** est l'**ID** du processus de LiveCode.

10.2 Le débogueur

D'habitude, quand vous voulez pour traquer un problème avec votre script, vous utilisez le débogueur. Le débogueur fournit une interface simple qui vous permet de parcourir votre script ligne par ligne lors de son exécution. Vous pouvez voir les résultats de chaque déclaration au fur et à mesure de son exécution. Si vous chargez l'observateur de variables depuis le débogueur, il vous montrera le contenu de toutes les variables utilisées dans un gestionnaire particulier. Ces valeurs seront mises à jour pas à pas. Vous pouvez même modifier ces valeurs pendant que votre script est exécuté. Cela peut être utile si vous repérez un problème dans un domaine donné ou voulez tester une section de code avec un ensemble différent de données pendant son exécution.

Pour activer le débogueur, vérifiez d'abord que **Script Debug Mode** est activé dans le menu **Development**. Ensuite, ouvrez le script que vous souhaitez déboguer et cliquez dans la barre grise à gauche de la ligne où vous voulez ouvrir le débogueur. Sinon, vous pouvez placer la commande **breakpoint** dans le script. L'utilisation de la commande **breakpoint** vous permet d'interrompre sous condition :

```
if tVar is true then breakPoint.
```

Ensuite, exécutez votre script normalement. Quand LiveCode atteint un point d'arrêt, il va suspendre l'exécution et afficher le débogueur.

Important : Pour voir le contenu de variables tandis que le script est en cours d'exécution, attendez l'ouverture du débogueur à ouvrir, puis choisir **Variable Watcher** dans le menu **Debug**.

10.3 Traitement des erreurs personnalisé

Si vous créez une application qui va être distribuée, vous voudrez peut-être inclure une méthode pour la capture et la gestion des erreurs. LiveCode fournit deux de ces méthodes.

La première est la structure de contrôle **try**/**catch**. Cette structure de contrôle peut être insérée dans n'importe quelle routine que vous savez pouvoir rencontrer des problèmes lors de l'utilisation. Par exemple, vous pouvez l'inclure dans une routine qui lit dans un fichier du disque de façon à gérer le cas où un fichier corrompu a été sélectionné.

La deuxième méthode consiste à écrire une routine **ErrorDialog** personnalisée. Contrairement à une structure de contrôle **try** / **catch**, un gestionnaire **ErrorDialog** est globale pour votre application (ou la carte ou la pile dans une application) et ne prévoit pas de mécanisme pour permettre à un script qui rencontre une erreur de pouvoir continuer. Utilisez cette méthode pour afficher une boîte de dialogue d'erreur personnalisée en cas d'erreur que vous êtes incapable de prévoir et de rapporter en utilisant **try** / **catch**.

Utilisation de try / catch

Insérez le code qui peut être source d'erreurs au sein d'une structure de contrôle **try**. L'exemple suivant montre une routine qui ouvre et lit un fichier joint dans une instruction **try**. Si une erreur d'exécution se produit dans l'une des déclarations après le début de la structure de contrôle **try**, la clause **catch** sera déclenchée avec les détails de l'erreur. Dans l'exemple ci-dessous, nous déclarons une variable **someError** de façon à contenir les détails de l'erreur :

```
try
  open file tFile
  read from file tFile until eof
  close file
catch someError
  answer "Une erreur s'est produite en lisant un fichier" && someError
end try
```

Astuce : Les données retournées à la routine d'erreur sont renvoyées dans le format interne que LiveCode utilise pour afficher les erreurs d'exécution. Pour consulter la chaîne de caractères associée à une erreur particulière sous une forme plus lisible, repérez le premier item retourné contre la liste des erreurs d'exécution stocké dans l'IDE LiveCode. Cela ne fonctionnera que dans l'IDE.

```
put line (item 1 of someError) of the cErrorsList of card 1 of stack "revErrorDisplay"
```

Si vous voulez inclure des déclarations qui seront exécutées indépendamment de savoir s'il y a eu une erreur ou non, inclure ces déclarations dans le cadre d'une clause **finally**.

Pour créer des messages d'erreur lisibles pour les cas où vous prévoyez qu'il peut y avoir une erreur, utilisez le mot clé **throw**. Par exemple, si nous voulons afficher un message d'erreur lorsque le résultat de l'ouverture d'un fichier retourne quelque chose :

```
open file tFile
if the result is not empty then throw the result
```

Dans l'exemple ci-dessus, si le fichier ne peut être ouvert et le résultat positif, la valeur du résultat sera transmise au relevé de capture dans la variable **someError**.

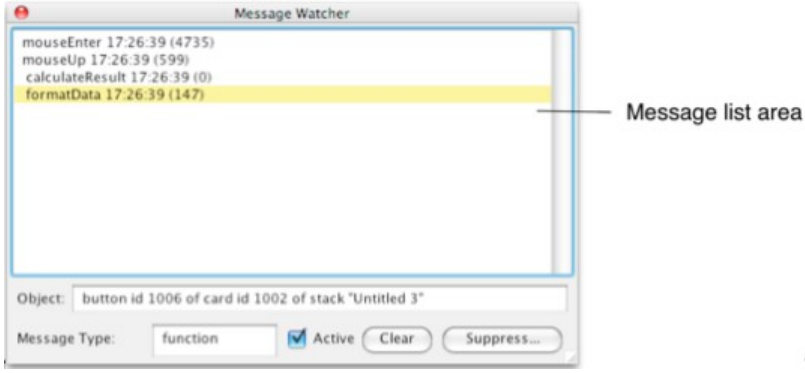
Ecrire Une Routine ErrorDialog Personnalisée

Lorsqu'une erreur d'exécution se produit, un message **ErrorDialog**, est envoyé. L'IDE utilise ce message pour afficher la boîte de dialogue d'erreur d'exécution. Cependant, vous pouvez écrire et inclure votre propre routine **ErrorDialog** personnalisée. Ceci est utile si vous envisagez de distribuer votre application. Par exemple, vous pouvez créer une pile qui transmet les informations d'erreur directement à votre service de support technique ou affiche un message personnalisé sur l'écran. Une routine de base **ErrorDialog** est indiqué ci-dessous :

```
on errorDialog pError
  answer "Il y a une erreur" && pError
end errorDialog
```

Cette routine est également activée si vous utilisez le mot clé **throw** pour générer une erreur (en dehors d'une structure de contrôle **try** / **catch**).

10.4 Message Watcher (observateur de messages)



L'observateur de message vous permet de voir quels messages sont envoyés au cours d'une opération particulière. Il peut être utile si vous voulez créer un journal des séquences de messages générés au cours d'une opération particulière. Il enregistre également la durée que chaque message prend pour exister. Ceci le rend utile lorsque vous recherchez des goulets d'étranglement dans votre code.

L'observateur de message se trouve dans le menu **Development**.

Zone de la liste des messages Listes les messages dès qu'ils se produisent. Le format est le nom du message, le moment où le message a été envoyé et le nombre de millisecondes depuis le dernier message. Cliquez pour sélectionner un message, double-cliquez pour l'ouvrir dans l'éditeur de code.

Dans l'exemple ci-dessus, nous pouvons voir que l'utilisateur est resté inactif pendant 4,7 secondes avant de passer la souris sur un objet qui a déclenché un message **mouseEnter**.
599 millisecondes plus tard il a cliqué la souris déclenchant un message **mouseUp**.
Le gestionnaire **mouseUp** a appelé **calculateResult** 0 millisecondes plus tard.
Ensuite **formatData** a été appelé 147 millisecondes plus tard. Parce qu'il n'y avait pas de message généré par l'utilisateur pendant la séquence, nous savons que le gestionnaire **calculateResult** a pris 147 millisecondes pour s'exécuter.

Object field Affiche l'objet à qui le message a été envoyé. Cliquez sur une ligne dans la liste des messages pour mettre à jour ce champ.

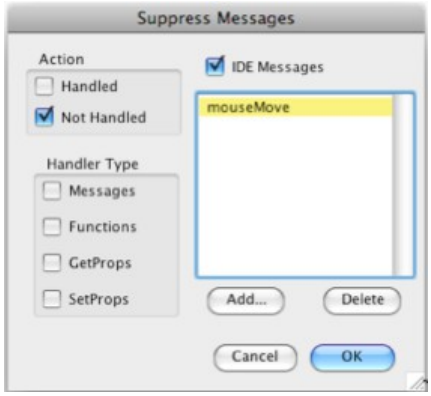
Message type Indique le type de message sélectionné - commande, fonction, **getProp** ou **setProp**

Active Cochez cette case pour activer l'observateur de messages. Désactivez-le lorsque vous avez obtenu des informations sur la séquence des événements qui vous intéressent et que vous souhaitez arrêter la journalisation des messages supplémentaires.

Clear Efface la liste dans le champ de message.

Suppress Permet de définir quels sont les messages que vous souhaitez afficher et ceux que vous souhaitez supprimer. Utilisez cette option pour affiner les messages qui sont enregistrés de sorte que vous pouvez voir des informations sur la séquence qui vous intéresse.

Voir ci-dessous pour plus d'informations.



Action – Handled Ne pas consigner n'importe quel message qui provoque l'exécution d'un gestionnaire, quand il est envoyé.

Action – Not Handled Ne pas consigner tout message qui ne provoque pas l' exécution d'un gestionnaire quand il est envoyé. C'est l'option par défaut et empêche le journal de se remplir de messages qui ne causent pas l'exécution de scripts.

IDE Messages Ne pas enregistrer les messages IDE de LiveCode. LiveCode IDE génère beaucoup de messages puisqu'il est écrit en LiveCode. C'est l'option par défaut, mais vous pouvez souhaiter afficher ces messages si vous personnalisez l'IDE LiveCode.

Handler Type Ne pas consigner le type choisi de gestionnaire. Par exemple, pour empêcher l'affichage de tous les appels de fonction, cochez la case fonction.

Message list Une liste des messages à ne pas consigner. Par défaut **mouseMove** est répertorié, sinon les messages **mouseMove** peuvent envahir l'écran lorsque vous déplacez la souris.

Add Delete Ajouter le nom d'un message pour l'empêcher d'être enregistré. Supprimer le nom du message pour l'amener à être enregistré dans le futur.

10.5 suivi des problèmes dans les applications autonomes

Nous vous recommandons de de déboguer votre application aussi complètement que possible dans l'IDE, de sorte que vous pouvez utiliser le débogueur et les autres outils. Mais parfois, vous pourriez avoir besoin pour traquer un problème qui ne se produit que dans une application autonome.

Dans ce cas, utiliser les techniques décrites ci-dessus pour l'écriture d'informations vers la sortie standard **stdOut** ou un fichier texte.

Vous pouvez également inclure une boîte de dialogue d'affichage des erreurs en utilisant l'onglet **Bug Reports** sur l'écran **Standalone Settings**. Assurez-vous de cocher la case ***Allow user to save report to file*** ou ***Allow user to email report***, afin que vous puissiez connaître l'erreur générée. Pour plus d'informations, voir le chapitre sur la distribution de votre application.

Chapitre 11 Transfert Informations : fichiers, Internet, Sockets

La lecture et l'écriture de données sur des fichiers ou transférer des données sur Internet sont des fonctions importantes dans la plupart des applications. LiveCode offre un riche ensemble de fonctionnalités pour effectuer ces opérations.

Accéder aux données d'un fichier prend généralement une seule ligne de code. La syntaxe du chemin du fichier dans LiveCode utilise le même format sur chaque plate-forme de sorte que vous n'avez généralement pas besoin de réécrire vos routines de manipulation de fichiers pour déployer en multiplateforme. Un ensemble de fonctions permet la copie, la suppression ou le renommage de fichiers, ainsi que l'accès au système approprié et aux dossiers d'utilisateur.

LiveCode comprend des fonctions pour le téléchargement et l'envoi de données vers l'Internet. De simples téléchargements et envois peuvent être réalisées avec une seule ligne de code. Les supports aux protocoles **http**, **ftp** et **PostX** sont inclus. La syntaxe est incluse permettant de télécharger tant en premier plan qu'en arrière-plan. Des commandes de bibliothèque supplémentaires vous permettent de construire des formulaire de données multiples, d'envoyer des commandes **ftp** et plus encore.

LiveCode comprend un support intégré pour **HTTPS**, **SSL** et le chiffrement.
Si la prise en charge intégrée du protocole ne fait pas ce dont vous avez besoin, LiveCode vous permet également de mettre en œuvre vos propres protocoles Internet en utilisant son support de prise (**socket**) directe. Une application client-serveur très basique peut être écrite en quelques lignes de code.

11.1 Spécifications des noms de fichiers et chemins de fichiers

Un chemin de fichier est une façon de décrire l'emplacement d'un fichier ou un dossier afin qu'il puisse être trouvé par un gestionnaire. Les chemins d'accès sont utilisés dans tout LiveCode : quand vous lisez et écrivez dans des fichiers texte, lorsque vous référencez un fichier QuickTime externe pour afficher dans un lecteur, et dans bien d'autres situations. Si votre application se réfère à des fichiers externes de quelque manière, une compréhension du chemin de fichier est indispensable.
Cette rubrique traite de la syntaxe pour créer et lire une référence de fichier, et comment relier les chemins de

fichiers à l'emplacement de votre application de sorte à ce qu'ils restent accessibles lorsque votre application est installée sur un autre système avec une structure de dossier différent.

11.1.1 Qu'est-ce qu'un chemin de fichier?

Un chemin de fichier est une description de l'emplacement exact d'un fichier ou d'un dossier. Le chemin du fichier est créé en commençant par la racine du système de fichiers de l'ordinateur, en nommant le disque ou le volume dans lequel le fichier est présent, ensuite nommer chaque dossier qui englobe le fichier, dans l'ordre, jusqu'à ce que le fichier soit atteint.

Localisation d'un fichier

Par exemple, supposons que vous voulez pour décrire l'emplacement d'un fichier appelé «Mon fichier», qui se trouve dans un dossier appelé "Mon dossier". Ce dossier est à son tour placé à l'intérieur d'un dossier appelé "Dossier Maître", qui se trouve sur un disque appelé "Disque Dur". Vous avez besoin de toutes ces informations pour décrire complètement où le fichier se trouve :

Disque Dur
Dossier Maître
Mon dossier
Mon fichier

Si quelqu'un vous demande dans quel disque le fichier est présent, ensuite le dossier à ouvrir, et ainsi de suite, vous pouvez trouver le fichier en ouvrant chaque icône successive sur le bureau de votre ordinateur. En démarrant depuis le disque, puis en ouvrant chaque dossier successivement jusqu'à arriver au fichier, vous pouvez trouver exactement le fichier décrit.

La structure d'un chemin de fichier

Un chemin d'accès spécifie chaque niveau de la hiérarchie qui englobe le fichier. LiveCode présente les informations dans un chemin de fichier qui pourrait ressembler à ceci :

/Disque dur/Dossier Maître /Mon dossier/Mon fichier

Astuce : Pour voir le chemin d'un fichier, entrez la commande suivante dans la boîte de message :

```
answer file "Choisir un fichier :"; put it
```

Cela affiche le chemin de fichier du fichier que vous choisissez.

Important : Chaque plate-forme a sa propre façon pour les programmeurs de spécifier les chemins de fichiers. Le chemin du fichier ci-dessus est dans le style habituel pour les chemins de fichiers sur les systèmes Unix. Pour assurer la compatibilité multi-plateforme, LiveCode utilise cette même barre oblique **/** dans son chemin du fichier quelle que soit la plateforme utilisée. De cette façon, vous pouvez généralement préciser un fichier et travailler avec les chemins dans vos scripts sans avoir à les convertir lorsque vous changez de platesforme.

Chemins de fichier sur les systèmes Windows

Sur les systèmes Windows, les disques sont nommés avec une lettre de lecteur suivie d'un caractère deux-points (:). Un chemin de fichier LiveCode typique sur un système Windows ressemble à ceci : C:/folder/file.txt

Chemins de fichier sur les systèmes OS X

Sur les systèmes OS X, le disque de démarrage, plutôt que le bureau, est utilisé comme le plus haut niveau de la structure du dossier. Cela signifie que le nom du disque de démarrage n'apparaît pas dans les chemins de fichiers. Au lieu de cela, la première partie du chemin du fichier est le dossier de niveau supérieur dans lequel le fichier est placé

Si le disque «Hard Disk» est le disque de démarrage, un parcours typique sur les systèmes OS X pourrait ressembler à ceci : /Top Folder/My Folder/My File

Notez que le nom du disque ne fait pas partie de ce chemin.

Pour les fichiers sur un disque qui n'est pas le disque de démarrage, le chemin du fichier commence par **"/Volumes"** au lieu de **"/"**. Un chemin de fichier typique d'un fichier qui se trouve sur un disque non-système sur un système OS X ressemble à ceci : /Volumes/Swap Disk/Folder/file.txt

Les chemins des dossiers

Note : Si vous devez trouver le nom du disque de démarrage, vérifiez le nom du premier disque renvoyé par la fonction **volumes.**

Vous construisez le chemin d'un dossier de la même façon que le chemin d'un fichier. Un chemin de dossier se termine toujours par une barre oblique **(/)**. Ce slash final indique que le chemin est un dossier plutôt qu'un fichier.
Par exemple, ce chemin décrit un dossier appelé **Project** dans un dossier appelé **Forbin** sur un disque nommé **Doomsday** : /Doomsday/Forbin/Project/

Si **Project** est un fichier, son chemin d'accès ressemble à ceci, sans le slash final: /Doomsday/Forbin/Project

Chemins de fichier pour paquets OS X

Un paquet est un type particulier de dossier, utilisé sur OS X, qui est présenté à l'utilisateur comme un seul fichier, mais qui est géré en interne par le système d'exploitation comme un dossier. De nombreuses applications OS X - y compris LiveCode et les applications qu'il crée - sont stockées et distribuées sous forme de paquets qui contiennent plusieurs fichiers.

Lorsque l'utilisateur double-clique sur le paquet, l'application démarre au lieu d'une fenêtre d'ouverture de dossier pour afficher le contenu du paquet.

Vous pouvez profiter du concept de paquet pour inclure tous les fichiers de support nécessaires à votre application. Si vous placez les fichiers dans le paquet de l'application, les utilisateurs ne les voient jamais d'ordinaire, et l'ensemble de l'application - les fichiers de support et le reste - se comporte comme une seule icône.

Déplacer, renommer ou supprimer un paquet

Lorsque vous utilisez la commande **rename** pour renommer un paquet, utilisez la forme **rename folder** de la commande:

```
rename folder "/Volumes/Disk/Applications/MyApp/" to "/Volumes/Disk/Applications/OtherApp/"
```

De même, lorsqu'il s'agit d'un bundle, utilisez la commande **delete Folder** plutôt que **delete file**, et la commande **revCopyFolder** plutôt que **revCopyFile**.

Se référer aux fichiers à l'intérieur d'un paquet

Astuce : Pour voir le contenu d'un paquet, faites un clic droit (ou contrôle clic) sur le paquet et choisissez "Afficher le Contenu du Paquet" dans le menu contextuel.

Lorsque vous faites référence à un fichier qui est à l'intérieur d'un paquet, vous pouvez traiter l'ensemble comme s'il s'agissait d'un dossier. Par exemple, si vous avez placé un fichier appelé "Mon Support.txt" à l'intérieur du paquet de votre application, le chemin absolu vers le fichier pourrait ressembler à ceci : /Volumes/Disk/Applications/MyApp/My Support.txt

11.1.2 Chemins de Fichiers Absolus et Relatifs

En décrivant comment arriver à un fichier, vous avez deux options. Vous pouvez commencer à la racine, le nom du disque, et de nommer chacun des dossiers contenants jusqu'à ce que vous obteniez le fichier. C'est ce qu'on appelle un chemin absolu, car il est indépendant de l'endroit d'où vous commencez à partir. Ou vous pouvez commencer à partir du dossier courant et décrire comment obtenir le fichier à partir de là. C'est ce qu'on appelle un chemin relatif, car il dépend de l'endroit où vous commencez.
Tous les chemins de fichier indiqués jusqu'ici dans cette rubrique sont les chemins absolus.

Chemins absolus

Les chemins absolus ne dépendent pas du dossier dans lequel le fichier de la pile se trouve, ou bien le lieu du dossier actuel. Un chemin absolu vers un dossier ou un fichier particulier est toujours écrit de la même façon. Par exemple, supposons que votre application se trouve dans un dossier appelé "Application Folder", et que vous souhaitez spécifier un fichier appelé "Westwind" qui est dans un dossier appelé "Stories" à l'intérieur de "Application Folder".

Hard Disk
- **Top Folder**
  - **My Folder**
    - **My File**
- **Application Folder**
  - **My Application**
  - **Stories**
    - **Westwind**

Le chemin absolu du fichier de votre application ressemble à ceci : /Hard Disk/Application Folder/My Application
et le chemin absolu du fichier "Westwindx" ressemble à ceci : /Hard Disk/Application Folder/Stories/Westwind

Chemins de fichiers relatifs

Supposons maintenant que vous voulez dire à quelqu'un comment se rendre au fichier "Westwind", à partir du dossier contenant l'application.

Puisque l'application est en "Application Folder", nous n'avons pas besoin de comprendre les étapes pour arriver à "Application Folder". Au lieu de cela, nous pouvons décrire l'emplacement du fichier "Westwind" avec ce chemin relatif : Stories/Westwind

Ce chemin relatif commence à "Application Folder" - le dossier qui contient l'application - et décrit comment obtenir le fichier "Westwind" à partir de là : vous ouvrez le dossier "Stories", puis trouvez "Westwind" à l'intérieur.

Un chemin de fichier relatif commence à un dossier particulier, plutôt que dans la partie supérieure du système de fichiers comme le chemin de fichier absolu. Le chemin de fichier relatif construit un chemin de fichier dans le dossier de départ pour le fichier ou le dossier dont l'emplacement est spécifié.

Trouver le dossier courant

Par défaut, le dossier en cours est réglé sur le dossier contenant l'application (soit l'environnement de développement de LiveCode ou bien votre application). Ainsi dans l'exemple ci-dessus, le dossier en cours est **Application Folder**, car c'est là que l'application en cours d'exécution se trouve.

En remontant vers le dossier parent

Le chemin relatif ".." indique le dossier parent du dossier actuel. Si le dossier actuel est **Stories**, le chemin relatif .. signifie la même chose que le chemin absolu : /Hard Disk/Application Folder/

En remontant plusieurs niveaux

Pour aller jusqu'à plus d'un niveau, utiliser plus d'un ".. /". Pour aller jusqu'à deux niveaux, utilisez ".. / .. /"; pour aller jusqu'à trois niveaux, utilisez ".. / .. / .. /", et ainsi de suite.

Remarque : Sur les systèmes OSX et Unix, les chemins absolus, commencent toujours par une barre oblique. Sur les systèmes Windows, les chemins d'accès absolus commencent toujours par une lettre de lecteur suivie de deux points (:).

Remarque : Pour modifier les dossier en cours, définissez la propriété **defaultFolder**.

Par exemple, supposons que le dossier courant est **Stories**, et son chemin absolu ressemble à ceci :
/Hard Disk/Application Folder/Stories/

Pour arriver à **My Application** dans **Application Folder**, vous montez d'un niveau à **Application Folder**, puis descendez un niveau plus bas vers **My Application**. Le chemin relatif ressemble à ceci : ../My Application

Pour arriver à "Top Folder" dans "Hard Disk", vous montez deux niveaux - vers "Application Folder", puis vers "Hard Disk" - et puis un niveau plus bas vers "Top Folder".

Le chemin relatif ressemble à ceci : ../../Top Folder/

À Partir du Répertoire Maison

Sur OS X et les systèmes Unix, le caractère "~" désigne le répertoire personnel de l'utilisateur. Un chemin qui commence par «~ /» est un chemin relatif à commencer par le répertoire personnel de l'utilisateur actuel. Un chemin qui commence par "~", suivie de l'**ID** utilisateur d'un utilisateur sur ce système, est un chemin relatif commençant par le répertoire personnel de l'utilisateur.

11.1.3 Quand utiliser les chemins d'accès relatifs et absolus

Les chemins de fichiers absolus et les chemins de fichiers relatifs sont interchangeables. Lequel utiliser dépend de quelques facteurs.

Les chemins absolus sont faciles à comprendre et ils ne changent pas en fonction du dossier en cours. Cela peut être un avantage si vous changez le **defaultFolder** régulièrement.
Les chemins de fichiers absolus incluent cependant toujours le nom complet du disque dur et les dossiers qui ont précédé le dossier de travail courant.

Par conséquent, si vous envisagez de distribuer votre application, vous devrez travailler avec des chemins relatifs, afin que que les médias embarqués dans les sous-dossiers avec votre application resten t toujours facile à localiser.
Astuce : Par défaut, en créant un lien vers une image ou une ressource à l'aide de l'inspecteur, LiveCode

insère un chemin de fichier absolu. Si vous envisagez de distribuer votre application, placez vos fichiers multimédias dans un sous-dossier à côté de la pile sur laquelle vous travaillez et convertissez ces chemins d'accès en chemins de fichiers relatifs par la suppression des répertoires jusqu'à celui dans lequel vous travaillez. Cela signifie que vous n'aurez pas besoin d'apporter des changements lorsque viendra le moment de distribuer votre application.

Il est correct d'utiliser des chemins absolus pour spécifier les fichiers ou les dossiers que l'utilisateur sélectionne après l'installation. Par exemple, si vous demandez à l'utilisateur de sélectionner un fichier (en utilisant la commande **answer file**) et de lire les données du fichier, il n'est pas nécessaire de convertir le chemin absolu que la commande **answer file** fournit vers une chemin relatif. Parce que vous utilisez le chemin juste après l'avoir obtenu à partir de la commande **answer**, vous savez que le nom du disque et de la structure du dossier ne vont pas changer entre l'obtention de du chemin et son utilisation.

11.2 Dossiers Spéciaux

Les systèmes d'exploitation modernes comportent chacun un ensemble de dossiers spécialisés désignés pour une variété de buts. Si vous écrivez une application, il est recommandé de faire usage de ces dossiers, le cas échéant, afin de fournir la meilleure expérience utilisateur possible. Par exemple, le contenu du bureau se trouve dans un dossier spécial ; il existe un dossier réservé pour les polices ; il y a un dossier pour les préférences de l'applications, et ainsi de suite.

Ces dossiers spéciaux n'ont pas toujours le même nom et emplacement, de sorte que vous ne pouvez pas compter sur un chemin de fichier enregistré pour les localiser. Par exemple, si votre application est installée sur un système d'exploitation localisé dans une autre langue, les noms des chemins de fichier seront différents. Certains dossiers spéciaux de Windows sont nommés ou placés différemment selon la version de Windows en cours d'exécution, etc

Pour connaître le nom et emplacement d'un dossier spécial, indépendamment de l'un de ces facteurs, vous utilisez la fonction **specialFolderPath**. La fonction prend en charge un certain nombre de dossiers pour chaque système d'exploitation, décrivant les dossiers spéciaux pour chacun. Certains des dossiers sont les même entre différentes plates-formes. L'exemple suivant va obtenir l'emplacement du dossier **Desktop** sur Windows, Mac OS X ou bien Linux:

```
put specialFolderPath("Desktop") into myPath
```

Pour obtenir le chemin d'accès au dossier du menu Démarrer sur un système Windows :

```
put specialFolderPath("Start") into myPath
```

Pour une liste complète des dossiers possibles, voir la fonction **specialFolderPath** dans le Dictionnaire du LiveCode.

11.3 Types de fichiers, Signatures d'application et Propriété des fichiers

Lorsque vous double-cliquez sur un fichier de document, il s'ouvre automatiquement dans l'application, il est associé. Chaque système d'exploitation a une méthode différente pour associer les fichiers avec une application. Afin de créer des fichiers qui appartiennent à votre application autonome, vous devez configurer l'association de manière appropriée pour chaque plate-forme que vous distribuez .
Cette rubrique décrit comment associer correctement votre application avec les fichiers qu'il crée.

11.3.1 Windows Extensions de Fichier et propriété

Quand un fichier est enregistré sur un système Windows, une extension de trois caractères est habituellement ajouté au nom du fichier. L'extension spécifie le format du fichier.

Pour déterminer quelle application doit se lancer lorsque l'utilisateur double clique sur un fichier,
Windows vérifie le registre Windows pour savoir quelle application il a lui-même enregistré comme propriétaire de l'extension du fichier. Chaque application peut ajouter des clés dans la base de registre pour identifier certaines extensions de fichiers comme lui appartenant.

Les applications qui ne possèdent pas de fichiers

Si votre application ne crée pas fichiers dont vous souhaitez que l'application soit propriétaire, vous n'avez pas besoin d'apporter des modifications dans le registre ou de spécifier toutes les extensions.

Les applications qui possèdent leurs propres fichiers

Si votre application crée des fichiers avec sa propre extension personnalisée, lorsque vous installez l'application, vous devez apporter des modifications à la base de registre Windows pour identifier l'extension comme appartenant à votre application.

Les programmes d'installation de Windows, feront ces changements de registre automatiquement pour vous. Vous pouvez également effectuer ces modifications du Registre à l'aide la fonction de **setRegistry**.

Installation des icônes personnalisées

Chaque fichier Windows peut afficher sa propre icône. Vous pouvez avoir des icônes distinctes pour votre application et les fichiers qu'elle détient. Icône fichiers doivent être stockés au format **.ico**.

Les icônes des applications personnalisées

Si vous souhaitez inclure une icône personnalisée pour votre application, utilisez l'option "**Application Icon**" du panneau Windows de la fenêtre **Standalone Application Settings** pour spécifier le fichier de l'icône. Lorsque vous générez l'application, l'icône sera incluse dans l'application. Pour plus d'informations, voir le chapitre sur le déploiement de votre application.

Icônes de fichier personnalisées

Pour inclure une icône personnalisée pour vos documents, utilisez l'option "**Document Icon**" du panneau Windows de la fenêtre **Standalone Application Settings** pour spécifier le fichier de l'icône. Lorsque vous générez l'application, l'icône sera incluse dans l'application.

Important : Pour que la bonne icône apparaisse sur les fichiers créés par votre application, l'extension du fichier doit être enregistrée dans le Registre de Windows.

Les extensions de fichier

Vous pouvez ajouter une extension au nom de tout fichier Windows. L'extension peut contenir des lettres A à Z, les chiffres 0-9, '(apostrophe), @, #, $,%, ^, &, (,), -, _, {}, `, ou bien ~.
Le Registre Windows associe les applications avec l'extension pour les fichiers qui la possèdent.

11.3.2 OS X Types de fichiers et Créateurs

Sur OS X chaque fichier possède une extension de fichier qui détermine quelle est l'application propriétaire. Toutefois, les systèmes OS X peuvent également utiliser la signature unique du créateur à quatre caractères et un type de fichier à quatre caractères utilisé sur Mac OS Classic (voir ci-dessous pour plus d'informations).

Les applications OS X stockent les informations d'association de fichiers dans un fichier de liste de propriétés,

dit **plist**. Le **plist** de chaque application est stocké comme faisant partie de son paquet d'application.

Les Applications ne Possédant pas de Fichiers

Pour attribuer votre signature de créateur unique, lors de la construction d'une application, entrez la signature sur le volet OS X de la fenêtre **Standalone Application Settings**.
LiveCode inclut automatiquement la signature du créateur dans le **plist** de l'application.

Les Applications qui Possèdent leurs Propres Fichiers

Si votre application crée des fichiers avec la signature du créateur de votre application, vous devriez inclure dans le **plist** de votre application une entrée pour chaque type de fichier que vous utilisez.

1. Une fois que vous avez construit votre application autonome, procédez comme suit pour ouvrir le fichier **plist** : Faites un clic droit sur votre paquet de l'application, accédez au dossier de contenu et ouvrez le fichier **Info.plist**. Si vous avez installé les outils de développement d'Apple, vous avez une application appelée **Property List Editor**, que vous pouvez utiliser pour faire des modifications dans le fichier plist. Sinon, vous pouvez éditer le fichier dans un éditeur de texte.

2. Repérez les informations relatives au type de document. Dans **Property List Editor**, développez le nœud "**Root**", puis développez le nœud "**CFBundleDocumentTypes**", puis développez le nœud "**0**". Dans un éditeur de texte, placez "**CFBundleDocumentTypes**". En dessous, notez les balises "**<array>**" et "**<dict>**". Les informations pour le premier type de document, est comprise entre "**<dict>**" et "**</ dict>**".

3. Entrez la description du fichier, qui est une courte phrase indiquant de quel type de fichier il s'agit. Dans **Property List Editor**, modifiez la valeur de "**CFBundleTypeName**" avec la description que vous souhaitez utiliser. Dans un éditeur de texte, placez "**CFBundleTypeName**" dans les informations du document. Ci-dessous est la description du fichier, enfermé entre "**<string>**" et "**</ string>**" :

   <string>LiveCode Stack</string>

   Modifier la description, placez celle que vous souhaitez utiliser.

   Important : Ne modifiez pas les balises (entre guillemets "<" et ">"). Changez que ce qui est entre elles.

4. Saisissez l'extension de fichier. Dans **Property List Editor**, développez "**CFBundleTypeExtensions**" et saisissez l'extension de fichier dans le noeud "**0**". Dans un éditeur de texte, placez "**CFBundleTypeExtensions**" dans les informations du document. Ci-dessous il est le prolongement, inclus dans "**<array>**" et les balises "**<string>**". Changez l'extension, placez celle que vous souhaitez utiliser.

5. Entrez le type de fichier à quatre caractères. Dans Property List Editor, développez "**CFBundleTypeOSTypes**" et entrez le type de fichier dans le nœud "**0**". Dans un éditeur de texte, placez "**CFBundleTypeOSTypes**" dans les informations du document. Ci-dessous, se trouve le type de fichier, inclus dans "**<array>**" et les balises "**<string>**". Modifier le type de fichier, placez celui que vous souhaitez utiliser.

Si le format de ce type de fichier est standard (tel que du texte brut), utilisez un type standard (tel que "TEXT"). Si le format appartient à votre application, utilisez un type de fichier personnalisé de votre choix.

Important : Apple se réserve tous les types de fichiers sans majuscules. Si vous utilisez un type de fichier personnalisé pour votre application, assurez-vous qu'il contient au moins une lettre majuscule.

Si vous voulez attribuer plus de types de fichiers à votre application, copiez / collez la partie du fichier **plist** dans " **CFBundleTypes**" entre "**<dict>**" et "**</dic**t>", y compris ces balises. Les noeud "**CFBundleTypes**" doit maintenant contenir deux nœuds »**<dict>**» et tous leurs contenus. Répétez les étapes précédentes pour chaque type de fichier différent votre application peut créer.

Création de fichiers

Lorsque votre application crée des fichiers, définissez la propriété **fileType** pour la signature créateur souhaitée et le type de fichier pour le nouveau fichier. (Pour les fichiers de la pile créés avec la commande de sauvegarde, utilisez la propriété **stackFileType** à la place.) Lors de la création des fichiers, l'application utilise la valeur actuelle de la propriété **fileType** ou **stackFileType** pour déterminer le créateur et le type de fichier que ce nouveau fichier devrait avoir.

Il est important de comprendre que la signature créateur d'un fichier détermine quelle est l'application qui se lance automatiquement lorsque vous double-cliquez sur le fichier, mais n'empêche pas d'autres applications d'être en mesure d'ouvrir ce fichier. Par exemple, si votre application crée les fichiers de type "texte", n'importe quel éditeur de texte peut ouvrir les fichiers.

Si votre application crée des fichiers de pile, et utilise le type de fichier "**RSTK**", alors LiveCode sera en mesure d'ouvrir les fichiers de pile, ainsi que votre application.

Les extensions de fichier

Vous pouvez ajouter une extension au nom de n'importe quel fichier OS X. Lorsque l'utilisateur double clique sur un fichier sans signature créateur, le système d'exploitation utilise l'extension pour déterminer l'application à utiliser pour ouvrir le fichier.
Le nom d'un paquet applicatif devrait se terminer avec l'extension ". App".

Note : les recommandations d'Apple pour déterminer le type de fichier et le créateur sur les systèmes OS X sont actuellement en pleine mutation. La méthode recommandée pour le moment est de mettre un type de fichier et la signature créateur, et joindre également une extension à la fin du nom de chaque fichier lors de sa création. Les extensions valides sur les systèmes OS X ont jusqu'à douze caractères, et peuvent inclure les lettres A à Z, les chiffres des 0-9, $,%, _, ou ~.

11.3.3 Mac OS Classic Types de Fichiers et Créateurs

Quand un fichier est enregistré sur un système Mac OS, la signature d'un créateur à quatre caractères est enregistrée avec lui. La signature créateur spécifie quelle application possède le fichier. Toutes les applications Mac OS et Mac OS X devraient avoir une signature créateur unique.
Les fichiers Mac OS ont un type de fichier séparé, long également de quatre caractères, qui spécifie le format du fichier. Le type de fichier est également utilisé pour déterminer quelles applications (autre que le propriétaire) peuvent travailler avec ce fichier. Le type de fichier de toutes les applications est "APPL".

11.3.4 Unix Extensions de Fichier

Les systèmes Unix n'ont pas une méthode globale requise pour spécifier le type d'un fichier, mais la plupart des fichiers sur un système Unix sont créés avec des extensions dans le nom du fichier, similaires aux extensions utilisées sur les systèmes Windows. Ces extensions peuvent être de n'importe quelle longueur et peuvent inclure tous les caractères (autre que /).

11.4 Utilisation d'URL

Une **URL** est un conteneur pour un fichier (ou une autre ressource), qui peut être soit sur le même système sur lequel l'application s'exécute sur, ou sur un autre système qui est accessible par l'intermédiaire d'Internet.

Cette rubrique présente les différents schémas d'**URL** implémentés dans LiveCode, comment créer et manipuler des fichiers en utilisant des **URL**, et comment transférer des données entre votre système et un serveur **FTP** ou **HTTP**.

Pour bien comprendre cette rubrique, vous devez savoir comment créer des objets et écrire des scripts courts, et de comprendre comment utiliser des variables pour stocker les données. Vous devez également avoir une compréhension de base de la façon dont l'Internet fonctionne.

11.4.1 Un aperçu des URL

Dans le langage de LiveCode, une **URL** est un conteneur pour un fichier ou un autre document, tel que la sortie d'un **CGI** sur un serveur Web. Les données d'une **URL** peuvent être sur le même système sur lequel l'application s'exécute sur, ou peuvent être sur un autre système.
Les **URL** dans LiveCode sont écrits comme les **URL** que vous voyez dans un navigateur. Vous utilisez le mot clé **URL** pour désigner une **URL**, contenants le nom de l'**URL** entre guillemets.

Exemples :

```
put field "Info" into URL "file:myfile.txt"
get URL "http://www.example.org/stuff/nonsense.html"
put URL "ftp://ftp.example.net/myfile" into field "Data"
```

11.4.2 URL Schémas

Un schéma d'**URL** est un type d'**URL**. LiveCode prend en charge cinq schémas d'**URL** avec le mot-clé **URL**: **http**, **ftp**, **file**, **binfile**, et (pour la rétrocompatibilité sur Mac OS Classic et OS X) **resfile**.
Les schémas **http** et **ftp** désignent des documents ou des répertoires qui se trouvent sur un autre système qui est accessible via l'Internet. **File** et **binfile** désignent des fichiers locaux.

11.4.3 Le schéma http

Une **URL http** désigne un document à partir d'un serveur Web :

```
put URL "http://www.example.org/home.htm" into field "Page"
```

Lorsque vous utilisez une **URL HTTP** dans une expression, LiveCode, télécharge l'**URL** depuis le serveur et substitue les données téléchargées à l'adresse **URL**.
Lorsque vous mettez quelque chose dans une **URL HTTP**, LiveCode envoie les données vers le serveur Web :

```
put field "Info" into URL "http://www.example.net/info.htm"
```

Note : Comme la plupart des serveurs web ne permettent pas le téléchargement par **HTTP**. Insérer un objet via une **URL HTTP**, généralement ne sera pas couronné de succès. Vérifiez auprès de l'administrateur du serveur pour savoir si vous pouvez utiliser le protocole **HTTP** pour envoyer des fichiers.

Pour plus de détails sur les **URL http**, voir l'entrée pour le mot-clé **http** dans le Dictionnaire du LiveCode.

11.4.4 Le schéma ftp

Une **URL FTP** désigne un fichier ou un répertoire sur un serveur **FTP**:

```
get URL "ftp://user:passwd@ftp.example.net/picture.jpg"
```

Lorsque vous utilisez une **URL FTP** dans une expression, LiveCode, télécharge l'**URL** depuis le serveur et substitue les données téléchargées à l'adresse **URL**.

Lorsque vous mettez quelque chose dans une **URL FTP**, LiveCode envoie les données vers le serveur **FTP** :

```
put image 10 into URL "ftp://user:passwd@ftp.example.net/picture.jpg"
```

Les serveurs **FTP** requièrent un nom d'utilisateur et mot de passe, que vous pouvez de spécifier dans l'**URL**. Si vous ne spécifiez pas un nom d'utilisateur et un mot de passe, LiveCode ajoute le nom d'utilisateur "**anonymous**" et un mot de passe factice automatiquement, conformément à la convention pour les serveurs **FTP** publics.

Note : "Uploader" vers un serveur **FTP** nécessite généralement un nom d'utilisateur et le mot de passe enregistré.

Pour plus de détails sur les **URL FTP**, voir l'entrée du mot **ftp** dans le Dictionnaire du LiveCode.

Répertoires sur un serveur FTP

Une **URL** qui se termine par une barre oblique **(/)** désigne un répertoire (plutôt qu'un fichier). Une **URL FTP** dans un répertoire aboutit à une liste du contenu du répertoire.

11.4.5 Le schéma de fichier

L'**URL** d'un fichier désigne un fichier de votre système :

```
put field "Stuff" into URL "file:/Disk/Folder/testfile"
```

Lorsque vous utilisez un fichier dans une expression, LiveCode obtient le contenu du fichier que vous désignez et le substitue à l'**URL**. L'exemple suivant met le contenu d'un fichier dans une variable :

```
put URL "file:monfichier.txt" into maVariable
```

Lorsque vous placez des données dans un fichier, LiveCode écrit les données dans le fichier :

```
put myVariable into URL "file:/Volumes/Backup/data"
```

Remarque : Comme pour les variables locales, si le fichier n'existe pas, y placer des données crée le fichier.

Pour créer une **URL** à partir d'un chemin de fichier que fournit LiveCode, utilisez l'opérateur **&** :

```
answer file "Please choose a file to get:" -- résultat dans la variable automatique : it
get URL ("file:" & it) -- le chemin du fichier est dans it
```

La syntaxe du chemin du fichier et le schéma de fichiers :

Le schéma de fichier utilise la même syntaxe de chemin de fichier utilisé ailleurs dans les déclarations de LiveCode. Vous pouvez utiliser les chemins absolus et les chemins relatifs dans l'**URL** de fichier.

Conversion des marqueurs de fin de ligne

Différents systèmes d'exploitation utilisent différents caractères pour marquer la fin d'une ligne. Mac OS X utilise un caractère de retour (ASCII 13), les systèmes Unix utilisent un caractère saut de ligne (ASCII 10), et

les systèmes Windows utilisent un retour suivi d'un saut de ligne.

Pour éviter tout problème lors du transport d'une pile entre plates-formes, LiveCode utilise toujours des sauts de ligne à l'intérieur lorsque vous utilisez un fichier comme un conteneur. LiveCode traduit au besoin entre les deux, le marqueur de fin de ligne de votre système, et le caractère de saut de ligne dans LiveCode.

Pour éviter cette traduction, utiliser le schéma **binfile** (voir ci-dessous).

11.4.6 Le schéma binfile

Une **URL binfile** désigne un fichier sur votre système qui contient des données binaires :

```
put URL "binfile:beachball.gif" into image "Beachball"
```

Lorsque vous utilisez une **URL binfile** dans une expression, LiveCode obtient le contenu du fichier que vous désignez et le substitue à l'**URL**. L'exemple suivant écrit le contenu d'un fichier dans une variable :

```
put URL "binfile:picture.png" into pictVar
```

Lorsque vous placez des données dans une **URL binfile**, LiveCode place les données dans le fichier :

```
put pictVar into URL "binfile:/Volumes/Backup/pict.png"
put image 1 into URL "binfile:/image.png"
```

Comme pour les variables locales, si le fichier n'existe pas, y placer des données crée le fichier.

Note : Le schéma **binfile** fonctionne comme le schéma de fichiers, sauf que LiveCode ne cherche pas à convertir les marqueurs de fin de ligne. C'est parce que les caractères de retour et saut de ligne peuvent être présents dans un fichier binaire, mais non destinés à marquer la fin de la ligne. La modification de ces caractères peut corrompre un fichier binaire, de sorte que le système **binfile** les laisse intacts.

11.4.7 Le schéma de resfile

Sur Mac OS Classic (et parfois sur les systèmes OS X), les fichiers peuvent consister soit en un **fork** de données ou une ressource **fork** ou les deux.

Le **fork** de ressources contient des ressources définies comme les icônes, les définitions de menus, boîtes de dialogue, les polices, et ainsi de suite. Une **URL resfile** désigne la resource fork d'un fichier OS X ou Mac OS :

```
put myBinaryData into URL "resfile:/Disk/Resources"
```

Lorsque vous utilisez une **URL resfile** dans une expression, LiveCode obtient la resource **fork** du fichier que vous désignez et la substitue à l'**URL**.

11.4.8 Manipulation des contenus de l'URL

Vous utilisez une **URL** comme n'importe quel autre contenant. Vous pouvez obtenir le contenu d'une **URL** ou utiliser son contenu dans n'importe quelle expression. Vous pouvez aussi mettre toutes les données dans une **URL**.

Les **URL http**, **ftp**, **binfile**, et **resfile** peuvent contenir des données binaires.
Les **URL http**, **ftp**, et **file** peuvent contenir du texte.

Le mot-clé URL

Pour spécifier un conteneur **URL**, vous utilisez le mot clé **URL** devant l'**URL**, qui peut utiliser n'importe lequel des cinq schémas décrits ci-dessus :

```
if URL "http://www.example.net/index.html" is not empty...
get URL "binfile:/Applications/Hover.app/data"
put 1+1 into URL "file:output.txt"
```

Le mot-clé **URL** indique à LiveCode que vous utilisez l'**URL** comme un conteneur.

Remarque : Certaines propriétés (telles que le nom de fichier d'un lecteur ou une image) vous permettent de spécifier une adresse **URL** comme valeur de la propriété. Veillez à ne pas inclure le mot-clé **URL** lors de la spécification de ces propriétés, parce que l'utilisation du mot-clé **URL** indique que vous traitez l'**URL** comme un conteneur. Si vous utilisez le mot-clé **URL** lorsque vous spécifiez une telle propriété, la propriété est définie sur le contenu de l'**URL**, et non l'**URL** elle-même, et ce n'est généralement pas ce qui est voulu.

Utilisation du contenu d'une adresse URL

Comme avec d'autres conteneurs, vous pouvez utiliser le contenu d'une **URL** en utilisant une référence à l'**URL** dans une expression. LiveCode va substituer le contenu de l'**URL** à la référence.

Si le schéma d'**URL** fait référence à un fichier local (les **URL file**, **binfile** ou **resfile**), LiveCode lit le contenu du fichier et le substitue à la référence **URL** dans l'expression :

```
answer URL "file:../My File" -- affiche le contenu du fichier
put URL "binfile:flowers.jpg" into maVariable
put URL "resfile:Icons" into URL "resfile:New Icons"
```

Si le schéma d'**URL** fait référence à un document sur un autre système (**http** ou **ftp** Les **URL**), LiveCode va télécharger automatiquement l'**URL** en substituant les données téléchargées à la référence **URL** :

```
answer URL "http://www.example.net/files/greeting.txt"
```

Remarque : Si le serveur renvoie un message d'erreur - par exemple, si le fichier que vous indiquez dans l'**URL** http n'existe pas - alors le message d'erreur remplace la référence d'**URL** dans l'expression.

Important : Lorsque vous utilisez une **URL FTP** ou **HTTP** dans une expression, le gestionnaire s'interrompt jusqu'à ce que LiveCode ait terminé le téléchargement de l'**URL**. Si vous ne voulez pas bloquer LiveCode lors de l'accès à ces ressources, utilisez la forme **load URL** de la commande (voir ci-dessous).

Mettre des Données dans une URL

Comme avec d'autres conteneurs, vous pouvez mettre des données dans une **URL**. Le résultat de ceci dépend si le schéma d'**URL** spécifie un fichier sur votre système (**file**, **binfile** ou **resfile**) ou sur un autre système (**http** ou **ftp**).
Si le schéma d'**URL** fait référence à un fichier local (**file**, **binfile** ou **resfile**), LiveCode écrit les données dans le fichier spécifié :

```
put field "My Text" into URL "file:storedtext.txt"
put image 1 into URL "binfile:picture.png"
```

Si le schéma d'**URL** fait référence à un document sur Internet (**http** ou **ftp**), LiveCode envoie les données à l'adresse:

```
put myVar into URL "ftp://me:pass@ftp.example.net/file.dat"
```

Note : Comme la plupart des serveurs web ne permettent pas le téléchargement par **HTTP**, ceci ne sera généralement pas réussi avec le schéma **http**.

Expressions Chunk et les URL

Comme d'autres conteneurs, les **URL** peuvent être utilisés avec des expressions **chunk** pour spécifier une partie de ce qu'il y a dans une **URL** - une ligne, un objet, un mot, ou un caractère. De cette façon, n'importe quel morceau d'une **URL** est comme un conteneur lui-même. Pour plus d'informations sur les expressions de **chunk**, voir le chapitre sur le traitement du texte et de données. Vous pouvez utiliser n'importe quel morceau d'une **URL** dans une expression, de la même façon que vous utilisez une **URL** complète :

```
get line 2 of URL "http://www.example.net/index.html"
put word 8 of URL "file:/Disk/Folder/myfile" into field 4
if char 1 of URL "ftp://ftp.example.org/test.jpg" is "0"...
```

Vous pouvez également spécifier des plages, et même un morceau à l'intérieur d'un autre :

```
put char 1 to 30 of URL "binfile:/marks.dat" into myVar
answer line 1 to 3 of URL "http://www.example.com/file"
```

Mettre des données dans un chunk

Si l'**URL** est local (c'est à dire si c'est un **file**, **binfile** ou **resfile**), vous pouvez mettre une valeur dans une partie de l'**URL** :

```
put it into char 7 of URL "binfile:/picture.gif"
put return after word 25 of URL "file:../datafile"
put field 3 into line 20 of URL "file:myfile.txt"
```

Vous pouvez aussi mettre une valeur dans un morceau d'une **URL FTP** ou **HTTP**. Parce qu'il impossible de ne charger q'une partie de fichier, LiveCode télécharge le fichier, fait le changement, puis renvoie le fichier sur le serveur.

Astuce : Cette méthode est inefficace si vous avez besoin de faire plusieurs changements. Dans ce cas, il est plus rapide d'abord mettre l'**URL** dans une variable, remplacer le bloc que vous souhaitez modifier, puis mettre la variable dans l'**URL** :

```
put URL "ftp://me:secret@ftp.example.net/file.txt" into myVar
put field "New Info" after line 7 of myVar
put field "More" into word 22 of line 3 of myVar
put myVar into URL "ftp://me:secret@ftp.example.net/file.txt"
```

Cela garantit que ce fichier ne soit téléchargé et renvoyé qu'une seule fois, peu importe le nombre modifications à faire.

11.4.9 Les URL et la mémoire

Les **URL**, contrairement à d'autres conteneurs, ne sont mis en mémoire que lorsque vous utilisez l'**URL** dans une déclaration. Les autres conteneurs - comme les variables, champs, boutons, et des images - sont normalement conservés en mémoire, donc y accéder leur n'augmente pas l'utilisation de mémoire.

Cela signifie que pour lire une **URL** ou placer une valeur dans un bloc d'**URL**, LiveCode charge tout le fichier en mémoire. Pour cette raison, vous devez être prudent lors de l'utilisation d'une **URL** pour se référer à n'importe quel fichier très volumineux.

Même lorsqu'on se réfère à un seul bloc dans une **URL**, LiveCode doit placer l'**URL** complète dans la mémoire. Une expression telle que :

```
line 347882 of URL "file:bigfile.txt"
```

peut être évaluée très lentement, voire ne pas fonctionner du tout, si la mémoire disponible est insuffisante. Si vous faites référence à un bloc d'une **URL FTP** ou **HTTP**, LiveCode doit télécharger l'ensemble du dossier pour trouver le bloc que vous spécifiez.

Astuce : Si vous avez besoin de lire et d'écrire de grandes quantités de données dans un fichier, ou de chercher à travers le contenu d'un fichier de grande taille sans charger tout le contenu dans la mémoire, utilisez les commandes **open file**, **read from file**, **seek** et **closefile** au lieu des commandes **URL**. Pour plus d'informations sur ces commandes, voir le Dictionnaire du LiveCode.

11.4.10 Suppression des URL

Vous supprimez une **URL** avec la commande **delete URL**.

Pour supprimer un fichier local, vous utilisez une **URL file** ou **binfile** :

```
delete URL "file:C:/My Programs/test.exe"
delete URL "binfile:../mytext.txt"
```

Ce n'est pas grave si le fichier contient des données binaires ou du texte, pour la suppression, ces schémas d'**URL** sont équivalents.

Astuce : Vous pouvez également utiliser la commande **delete file** de supprimer un fichier.

Pour supprimer le **resource fork** d'un fichier, vous utilisez une **URL resfile**. L'exemple suivant supprime le **resource fork** avec toutes les ressources, mais laisse le fichier en place :

```
delete URL "resfile:/Volumes/Backup/proj.rev"
```

Astuce : Pour supprimer une ressource unique plutôt que la totalité **resource fork**, utilisez la fonction **deleteResource**.

Pour supprimer un fichier ou un répertoire sur un serveur **FTP**, vous devez utiliser l'**URL ftp**:

```
delete URL "ftp://root:secret@ftp.example.org/deleteme.txt"
delete URL "ftp://me:mine@ftp.example.net/trash/"
```

Comme pour la création de fichiers, vous pouvez utiliser une **URL http** pour supprimer un fichier, mais la plupart des serveurs **HTTP** ne sont pas configurés pour permettre cela.

11.5 téléchargement et téléversement des fichiers

Le plus simple pour transférer des données vers un serveur **FTP** ou **HTTP** est d'utiliser la commande **put** pour téléverser ou utiliser l'**URL** dans une expression à télécharger.

La bibliothèque Internet intègre des commandes supplémentaires pour téléverser et télécharger des fichiers depuis et vers un serveur **FTP**. Ces commandes offrent des options plus souples pour surveiller et contrôler la progression du transfert de fichier.

"Uploader" à l'aide la commande put

Comme mentionné ci-dessus, mettre un objet dans une **URL FTP** ou **HTTP**, envoie les données vers le serveur :

```
put myVariable into URL "ftp://user:pass@ftp.example.org/newfile.txt"
```

Si vous utilisez la commande d'une **URL file** ou **binfile** en tant que source, le fichier est déposé :

```
put URL "file:newfile.txt" into URL "ftp://user:pass@ftp.example.org/newfile.txt"
```

Lorsque vous téléchargez des données de cette façon, l'opération se bloque : c'est que le gestionnaire s'arrête jusqu'à ce que le transfert soit terminé. (Voir ci-dessous pour plus de détails sur la façon de créer un transfert de fichier qui ne soit pas de blocage.)

S'il y a une erreur, l'erreur est placée dans la fonction **result** :

```
put field "Data" into URL myFTPDestination
if the result is not empty then beep 2
```

Important : le téléchargement ou transfert d'une **URL** n'empêche pas d'autres messages d'être envoyés lors du transfert de fichiers : le gestionnaire actuel est bloqué, mais les autres gestionnaires ne le sont pas. Par exemple, l'utilisateur peut cliquer sur un bouton qui téléverse ou télécharge une autre **URL** alors que la première **URL** est toujours en cours de chargement. Dans ce cas, le deuxième transfert de fichier n'est pas effectué et le résultat est mis à "**Error Previous request has not completed**." Pour éviter ce problème, vous pouvez définir un indicateur tandis qu'une **URL** est téléversée, et vérifier ce drapeau lorsque vous essayez d'envoyer ou de télécharger des **URL**, pour vous assurer qu'il n'y ait pas déjà un transfert de fichier en cours.

Téléchargement à l'aide d'une URL

Faire référence à une **URL ftp** ou **http** dans une expression télécharge le document.

```
put URL "ftp://ftp.example.net/myfile.jpg" into image 1
get URL "http://www.example.com/newstuff/newfile.html"
```

Si vous utilisez la commande d'une **URL file** ou **binfile** comme destination, le document est téléchargé dans le dossier :

```
put URL "ftp://ftp.example.net/myfile.jpg" into URL "binfile:/Disk/Folder/myfile.jpg"
```

11.5.1 transferts non bloquants

Lorsque vous transférez un fichier en utilisant des conteneurs **URL** le transfert de fichiers arrête le gestionnaire actuel jusqu'à ce que le transfert s'effectue. Ce type d'opération est appelée une opération de blocage, car il y blocage du gestionnaire actuel.

Si vous souhaitez transférer des données en utilisant **http** sans blocage, utilisez la commande **load**. Si vous voulez transférer de gros fichiers via **FTP**, utilisez les commandes: **libURLftpUpload**, **libURLftpUploadFile** ou **libURLDownloadToFile**.

La non-blocage des transferts de fichiers a plusieurs avantages :

- Puisque contacter un serveur peut prendre un certain temps en raison de faiblesse du réseau, la pause impliquée dans une opération de blocage peut être assez longue pour être perceptible par l'utilisateur.
- Si une opération de blocage impliquant une **URL** est en cours, aucune autre opération de bloquante ne peut commencer tant que la précédente n'es pas terminée.
- Si un transfert de fichier non bloquant est en cours, vous pouvez lancer d'autres transferts de fichiers non-bloquants. Cela signifie que si vous utilisez les commandes de bibliothèque, l'utilisateur peut commencer de multiples transferts de fichiers sans erreurs.
- Lors d'un transfert de fichier non-bloquant, vous pouvez vérifier et visualiser l'état du transfert. Ceci vous permet d'afficher la progression du transfert et autorise l'utilisateur à annuler le transfert de fichier.

Utilisation de la commande load

La commande **load** télécharge le document spécifié dans en tâche de fond et le place dans un cache. Une fois qu'un document a été mis en cache, il est possible d'y accéder presque instantanément lorsque vous utilisez son **URL**. LiveCode utilise la copie en mémoire cache au lieu de télécharger à nouveau l'**URL**.

Pour utiliser un fichier qui a été téléchargé par la commande **load**, reportez-vous à l'utilisation du mot-clé **URL** comme d'habitude. Lorsque vous demanderez l'**URL** d'origine, LiveCode utilisera automatiquement le fichier cache.

Pour de meilleures performances, utilisez la commande **load** à un moment où la vitesse de réponse n'est pas critique (pas lorsque votre application est en cours de démarrage, par exemple), et ne l'utiliser que pour les documents qui doivent être affichés rapidement, comme des images du web qui seront visualisées quand vous accéderez à la prochaine carte.

Vérification de l'état lors de l'utilisation de la commande load

Même si un fichier est transféré en utilisant les commandes de chargement, vous pouvez vérifier l'état du transfert en utilisant la fonction **URLStatus**. Cette fonction retourne l'état actuel d'une **URL** qui est en cours de téléchargement ou téléversée :

```
put the URLStatus of "ftp://ftp.example.com/myfile.txt" into field "Current Status"
```

La fonction **URLStatus** retourne une des valeurs suivantes :

- *queued* - en attente jusqu'à ce qu'une demande précédente sur le même site soit terminée
- *contacted* - le site a été contacté mais aucune donnée n'a encore été envoyée ou reçue

- *requested* - l'**URL** a été demandée
- *loading bytesTotal*, *bytesReceived* - les données d'**URL** sont en train d'être reçues
- *uploading bytesTotal, bytesReceived* - le fichier est envoyé à l'adresse **URL**
- *cached* - l'**URL** est dans le cache et le téléchargement est terminé
- *uploaded* - l'application n'a fini de télécharger le fichier vers l'**URL**
- *error* - une erreur s'est produite et l'**URL** n'a pas été transférée
- *timeout* - l'application a expiré lors d'une tentative de transfert de l'**URL**

Pour surveiller la progression d'un transfert de fichier ou afficher une barre de progression, vérifiez en réitérant plusieurs fois la fonction **URLStatus** pendant le transfert. La meilleure façon de le faire est d'utiliser le message basé sur le timer - voir la section du même nom dans le chapitre Codage dans LiveCode, pour plus d'informations.

Annuler un transfert de fichiers et vider le cache

Pour annuler un transfert initié avec la commande **load** et vider le cache, utilisez la commande **unload**.

```
unload URL "http://example.org/new_beta"
```

Le téléchargement de gros fichiers via FTP

La bibliothèque Internet fournit un certain nombre de commandes pour le transfert de gros fichiers via **FTP** sans blocage.

- **libURLftpUpload** télécharge les données vers un serveur **ftp**
- **libURLftpUploadFile** Charge un fichier sur un serveur **ftp**
- **libURLDownloadToFile** télécharge un fichier d'un serveur **ftp** vers un fichier local

L'effet de base de ces commandes est le même que celui d'utiliser des **URL** : c'est à dire les données sont transférées vers ou depuis le serveur.
Cependant, il y a plusieurs différences dans la façon dont le transfert de fichier réel est réalisé. En raison de ces différences, les commandes de bibliothèque seront les plus appropriées pour les chargements et téléchargements, particulièrement si le fichier transféré est important.

Les jeux suivants de déclarations montre :

1. chacune des commandes de la bibliothèque Internet
2. avec l'emploi de l'équivalent **URL** :

```
1. libURLftpUpload myVar,"ftp://me:pass@example.net/file.txt"
2. put myVar into URL "ftp://me:pass@example.net/file.txt"
```

```
1. libURLftpUploadFile "test.data","ftp://ftp.example.org/test"
2. put URL "binfile:test.data" into URL "ftp://ftp.example.org/test"
```

```
1. libURLDownloadToFile"ftp://example.org/new_beta","/HD/File"
2. put URL "ftp://example.org/new_beta"into URL "binfile:/HD/File"
```

En utilisant des messages de rappel

Lorsque vous démarrez un transfert de fichier en utilisant la commande : **libURLftpUpload libURLftpUploadFile** ou **libURLDownloadToFile** vous pouvez éventuellement spécifier un message de rappel, qui est habituellement un message personnalisé pour lequel vous écrivez un gestionnaire dédié.
Ce message est envoyé lors des changements **URLStatus** de transfert de fichier.
Vous pouvez donc gérer le message de rappel pour gérer les erreurs ou pour afficher l'état du transfert de fichiers à l'utilisateur.

L'exemple simple suivant montre comment afficher un message d'état destiné à l'utilisateur.
Les deux gestionnaires suivants peuvent se trouver dans le script d'un bouton :

```
on mouseUp
  libURLDownloadToFile "ftp://example.org/new_beta","/HD/Latest Beta","showStatus"
end mouseUp
on showStatus theURL
  put the URLStatus of theURL into field "Status"
end showStatus
```

Lorsque vous cliquez sur le bouton, le gestionnaire **mouseUp** est exécuté. La commande **libURLDownloadToFile** commence le transfert de fichiers, et son dernier paramètre spécifie qu'un message **showStatus** sera envoyé sur le bouton à chaque modification de **URLStatus**.

Comme **URLStatus** change périodiquement tout au long du processus de téléchargement, le gestionnaire **showStatus** du bouton, est exécuté à maintes reprises.

Chaque fois qu'un message **showStatus** est envoyé, le gestionnaire met le nouveau statut dans un champ. L'utilisateur peut vérifier ce champ à tout moment pendant le transfert de fichiers pour voir si le téléchargement a commencé, combien de fichiers ont été transférés, et s'il y a eu une erreur.
Si un transfert de fichier a été lancé avec la commande: **libURLftpUpload**, **libURLftpUploadFile** ou **libURLDownloadToFile**, vous pouvez annuler le transfert en utilisant la commande **unload**.

Téléverser, Téléchargement et Mémoire

Lorsque vous utilisez une **URL** comme un conteneur LiveCode met l'**URL** complète en mémoire. Par exemple, si vous téléchargez un fichier depuis un serveur FTP en utilisant la commande **put** LiveCode télécharge tout le contenu du fichier en mémoire avant de le mettre dans le conteneur de destination. Si le fichier est trop volumineux pour tenir dans la mémoire disponible, le transfert de ce fichier en utilisant cette méthode échouera (et peut entraîner d'autres résultats inattendus).

Les commandes de bibliothèque : **libURLftpUpload**, **libURLftpUploadFile** et **libURLDownloadToFile** toutefois ne nécessitent pas que l'intégralité du fichier soit chargé en mémoire.
Au lieu de cela, elles transfèrent le fichier d'un seul tenant.
Si un fichier est (ou pourrait être) trop volumineux pour tenir confortablement dans la mémoire disponible, vous devriez toujours utiliser les commandes de bibliothèque pour le transférer.

11.5.2 L'utilisation d'une pile sur un serveur

Normalement, vous utilisez des fichiers de pile qui se trouvent sur un disque local. Vous pouvez également ouvrir et utiliser une pile qui se trouve sur un serveur **FTP** ou **HTTP**.
Grâce à cette fonctionnalité, vous pouvez mettre à jour une application en téléchargeant de nouvelles piles, donner de nouvelles fonctionnalités accessibles via Internet, et même conserver la majeure partie votre application sur un serveur plutôt que tout stocker localement.

Aller à une pile sur un serveur : comme pour les fichiers de pile locaux, vous utilisez la commande **go** pour ouvrir une pile stockée sur un serveur:

```
go stack URL "http://www.example.org/myapp/main.rev"
go stack URL "ftp://user:pass@example.net/secret.rev"
```

Note : Pour qu'une telle déclaration fonctionne, le dossier de la pile doit avoir été transféré sous forme de données binaires, non compressé, et ne pas utiliser un encodage de type **BinHex**.

Astuce : Si vous avez besoin de télécharger une grosse pile, utilisez la commande **load** pour effectuer le téléchargement avant d'utiliser la commande **go** pour afficher la pile. Cela vous permet d'afficher une barre de progression pendant le téléchargement.

LiveCode télécharge automatiquement le fichier pile. La pile principale du fichier de pile s'ouvre alors dans une fenêtre, comme si vous aviez utilisé la commande **go** pour ouvrir pour ouvrir un fichier local de pile.

Vous pouvez aller directement à une carte spécifique dans la pile :

```
go card "My Card" of stack URL "http://www.example.org/myapp/main.rev"
```

Pour ouvrir un sous-pile à la place, utiliser le nom de la sous-pile :

```
go stack "My Substack" of URL "http://www.example.org/myapp/main.rev"
```

Utilisation d'une pile compressée

Vous ne pouvez pas ouvrir directement une pile qui est compressée. Toutefois, étant donné que l'**URL** de la pile est un conteneur, vous pouvez utiliser l'**URL** comme paramètre de la fonction **decompress**. Celle-ci prend les données du fichier pile et les décompresse, restituant les données du fichier de pile initial. Vous pouvez ouvrir la sortie de la fonction directement comme si c'était une pile. La déclaration suivante ouvre un fichier pile compressé sur un serveur :

```
go decompress(stack URL "http://www.example.net/comp.gz")
```

La déclaration télécharge automatiquement le fichier "comp.gz", le décompresse, et ouvre la pile principale du fichier.

Sauvegarde des piles depuis un serveur

Quand une pile est téléchargée en utilisant la commande **go**, elle est chargée en mémoire, mais pas sauvegardée sur un disque local.
Une telle pile se comporte comme une nouvelle pile (non enregistrée) jusqu'à ce que vous utilisiez la commande **save** pour l'enregistrer comme un fichier de pile.

Nota : L'enregistrement d'une pile qui a été téléchargée avec la commande **go** n'est pas re-transférée sur son serveur. Pour transférer une pile modifiée, vous devez l'enregistrer dans un fichier local, puis utiliser l'une des méthodes décrites dans cette rubrique pour envoyer le fichier sur le serveur.

11.6 Autres commandes Internet

La bibliothèque Internet a un certain nombre de commandes supplémentaires pour travailler avec les formulaires Web, les commandes **ftp**, les réglages personnalisés et le dépannage. Ces commandes sont documentées plus en détail le Dictionnaire LiveCode.

Lancement du navigateur de l'utilisateur avec une adresse URL

Pour lancer le navigateur par défaut avec une **URL**, utilisez la commande **launch URL**.

```
launch URL "http://www.runrev.com/"
```

Remarque : Pour afficher les pages web dans LiveCode, au lieu de lancer un navigateur externe, utilisez **revBrowser**. Voir la section sur **revBrowser** pour plus d'informations.

11.6.1 Travailler avec les formulaires Web

Pour publier des données sur un formulaire web, utiliser les commande **post**. Pour encoder les données pour les rendre appropriées pour le **post**, utilisez la fonction **libURLFormData**. Pour créer des données de formulaires en plusieurs parties (comme décrit dans le RFC 1867) utiliser la fonction **libURLMultipartFormData**. Pour ajouter des données vers une formulaire multiple, un seul élément à la fois, utilisez la fonction **libURLMultipartFormAddPart**. Cela peut être utile si vous avez besoin de spécifier le type **mime** ou le codage de transfert pour chaque élément.

11.6.2 Travailler avec FTP

Pour plus de détails de base sur l'envoi et le téléchargement par **FTP**, consultez la section ci-dessus.

Les commandes suivantes fournissent des fonctionnalités supplémentaires lorsque l'on travaille avec le protocole **FTP** :

- **libURLSetFTPStopTime** - Définit le délai d'attente pour les transferts **FTP**.
- **libURLSetFTPMode** - alterner entre les modes actif et passif pour les transferts **FTP**.
- **libURLSetFTPListCommand** - Bascule entre l'envoi de formats **LIST** ou **NLST** en listant le contenu d'un répertoire **FTP**.
- **libURLftpCommand** – envoie une commande **FTP** à un serveur **FTP**.
- **libURLftpUpload** – envoie les données. Voir la section ci-dessus pour plus de détails.
- **libURLftpUploadFile** – envoie un fichier, sans charger tout le fichier en mémoire. Voir la section ci-dessus pour plus de détails.
- **libURLDownloadToFile** – télécharge les données vers un fichier, sans avoir à charger la totalité des données en mémoire. Voir la section ci-dessus pour plus de détails.

11.6.3 Méthodes HTTP et URL http

Les opérations de base utilisées par le protocole **HTTP** sont appelées méthodes. Pour les **URL http**, les méthodes **HTTP** suivantes sont utilisées dans les cas suivants :

- **GET** : quand une **URL http** est évaluée dans une expression
- **PUT** : lorsque vous placez une valeur dans une **URL http**
- **POST** : lorsque vous utilisez la commande **post**
- **DELETE** : lorsque vous utilisez la commande **URL delete** avec une **URL http**

Note : De nombreux serveurs **HTTP** n'appliquent pas la méthodes **PUT** et **DELETE**, ce qui signifie que vous ne pouvez pas mettre des valeurs dans une **URL http** ou supprimer une **URL http** sur ces serveurs. Il est courant d'utiliser le protocole **FTP** à la place pour charger et supprimer des fichiers, vérifiez avec l'administrateur de votre serveur pour savoir quelles méthodes sont supportées.

En-têtes HTTP

Quand LiveCode, délivre une requête **GET** ou **POST**, il construit un ensemble minimal d'en-têtes **HTTP**. Par exemple, lors de l'émission sur un système Mac OS, de la déclaration suivante:

```
put URL "http://www.example.org/myfile" into myVariable
```

se traduit dans l'envoi d'une requête **GET** au serveur :

GET /myfile HTTP/1.1
Host: 127.0.0.0
User-Agent: LiveCode (MacOS)

Vous pouvez ajouter des en-têtes ou remplacer l'en-tête d'hôte ou le **User-Agent**, en définissant la propriété **httpHeaders** avant d'utiliser **l'URL** suivante:

```
set the HTTPHeaders to "User-Agent: MyApp" & return & "Connection: close"
put URL "http://www.example.org/myfile" into myVariable
```

Maintenant la requête envoyée au serveur ressemble à ceci :

GET /myfile HTTP/1.1

Host: 127.0.0.0

User-Agent: MyApp

Connection: close

Le schéma d'**URL ftp** peut être utilisé pour créer un nouveau fichier sur un serveur **FTP**. Comme pour les schémas **file** et **binfile**, mettre un objet dans l'**URL** crée le fichier :

```
put dataToUpload into URL "ftp://jane:pass@ftp.example.com/newfile.dat"
```

Astuce : Vous pouvez créer un répertoire **FTP** en téléchargeant un fichier vers le nouveau répertoire (inexistant). Le répertoire est automatiquement créé. Vous pouvez ensuite supprimer le fichier, si vous le souhaitez, en laissant un nouveau répertoire vide sur le serveur :

```
put empty into URL "ftp://jane:pass@example.com/newdir/dummy" –Crée un fichier vide dans un dossier inexistant
delete URL "ftp://jane:pass@example.com/newdir/dummy" – Effacer le fichier pour laisser le nouveau dossier
```

11.6.4 Paramètres de transmission supplémentaires

Les commandes suivantes fournissent des options de personnalisation supplémentaires pour la bibliothèque Internet :

- **libURLSetExpect100** – Permet de définir une limite à la taille des données étant affichées avant de demander une réponse continue à partir du serveur.
- **libURLSetCustomHTTPHeaders** - Définit les en-têtes à transmettre à chaque requête au serveur **HTTP**. Voir aussi la section sur httpHeaders ci-dessus.
- **libURLFollowHttpRedirects** – Spécifie que les demandes **GET** devraient suivre les redirections **HTTP** et obtenir la page redirigée.
- **libURLSetAuthCallback** - Définit une fonction de rappel pour gérer l'authentification avec les serveurs **HTTP** et **proxy**.

11.6.5 Dépannage

Les commandes et les fonctions suivantes peuvent être utiles lors du débogage d'une application qui utilise la bibliothèque Internet.

- **resetAll** - Ferme tous les **sockets** ouverts, et arrête toutes les activités Internet en suspens.
  Attention : La commande **resetAll** ferme tous les **sockets** ouverts, ce qui inclut tous les autres **sockets** ouverts par l'application et toutes les **sockets** utilisés pour d'autres chargements et téléchargements. Pour cette raison, vous devez éviter l'utilisation systématique de la commande **resetAll**. Envisager de l'utiliser seulement au cours du développement, pour éclaircir des problèmes de connexion pendant le débogage.
- **libURLErrorData** - Retourne une erreur qui a été causée lors d'un téléchargement lancé avec la commande load.
- **libURLVersion** – Retourne la version de la bibliothèque Internet.
- **libURLSetLogField** – Spécifie un champ pour les informations de connexion pour les chargements et les téléchargements à l'écran.
- **libURLLastRHHeaders** - Retourne les en-têtes envoyés par l'hôte distant dans la transaction la plus récente **HTTP**.
- **libURLLastHTTPHeaders** - Retourne la valeur de **httpHeadersproperty** utilisée pour la requête HTTP précédente.

11.7 revBrowser - rendu d'une page Web dans une pile

Utiliser les commandes de **revBrowser** pour rendre une page Web dans un pile. **RevBrowser** utilise WebKit (Safari) sur Mac OS X et Internet Explorer sur Windows. Actuellement **RevBrowser** n'est pas supporté sous Linux.

Pour créer un objet du navigateur dans une pile, utilisez la fonction **revBrowserOpen**. Cette fonction utilise le **windowID** pour la pile dans laquelle voulez ouvrir le navigateur et une **URL**. Veuillez noter que le **windowID** n'est pas la même chose que la propriété **ID** de la pile.

```
put the windowid of this stack into tWinID
put revBrowserOpen(tWinID,"http://www.google.com") into sBrowserId
```

Pour définir des propriétés sur le navigateur, utilisez la commande **revBrowserSet**. Les commandes suivantes rendent la bordure visible et définit ensuite le rectangle pour être identique à une image nommée "**browserimage**" :

```
revBrowserSet sBrowserId, "showborder","true"
revBrowserSet sBrowserId, "rect",rect of img "browserimage"
```

Pour fermer le navigateur quand vous avez fini avec cela, utilisez la commande **revBrowserClose**. Cette commande utilise le **windowID** pour la pile contenant le navigateur :

```
revBrowserClose sBrowserId
```

RevBrowser prend en charge un certain nombre de paramètres et de messages. Vous pouvez intercepter un message chaque fois que l'utilisateur accède à un lien, bloquer la navigation, - intercepter les clics dans le navigateur, les demandes de téléchargement de fichiers ou pour ouvrir une nouvelle fenêtre.

Pour une liste complète des commandes qui opèrent sur **RevBrowser**, ouvrez le dictionnaire de LiveCode et tapez **browser** dans la boîte du filtre.

11.8 SSL et le cryptage

LiveCode inclut le support pour l'utilisation de **Secure Sockets Layer** et le protocole **https**. Il comprend également une bibliothèque de chiffrement de niveau industriel que vous pouvez utiliser pour crypter des fichiers ou des transmissions de données.

11.8.1 Chiffrer et déchiffrer des données

Pour crypter des données, utilisez la commande **encrypt**. La commande **encrypt** prend en charge une grande variété de méthodes standards de l'industrie de cryptage. La liste des méthodes installées peut être récupérée en utilisant la fonction **cipherNames**. Pour décrypter les données, utilisez la commande **decrypt**. Pour plus d'informations sur ces fonctionnalités, consultez le dictionnaire de LiveCode.

Astuce : Si vous utilisez la bibliothèque de chiffrement sur un système Windows, il est possible qu'une autre application ait installé des **DLL** qui utilisent le même nom que ceux qui sont inclus avec LiveCode, en charge du chiffrement. Vous pouvez forcer votre application à charger des **DLL SSL** de LiveCode, en définissant la variable d'environnement **$ PATH**, avant de charger la bibliothèque :

```
put $PATH into tOldPath
put <path to SSL DLLs> into $PATH
get the cipherNames -- Force le chargement des DLLs SSL
put tOldPath into $PATH
```

11.8.2 Connexion via HTTPS

Vous pouvez connecter et télécharger des données à partir d'une **URL** en **https** de la même façon que vous accédez à une **URL http**.

```
put URL "https://www.example.com/store.php"
```

S'il ya une erreur, elle sera placée dans le résultat. Si vous devez inclure un nom d'utilisateur et un mot de passe, vous pouvez le faire sous la forme suivante :

```
put URL "https://user:password@www.example.com/"
```

11.8.3 Mise en œuvre vos propres protocoles sécurisés

Pour implémenter votre propre protocole sécurisé, utilisez la variante **open secure socket** de la commande **open socket**. Vous pouvez spécifier s'il faut ou non inclure la certification, un certificat et une clé. Pour plus d'informations sur la commande **opens socket**, voir le dictionnaire de LiveCode.

11.9 Ecrire votre propre protocole avec les sockets

Si vous avez besoin de mettre en œuvre votre propre protocole, vous pouvez le faire en utilisant le support socket de LiveCode. Pour comprendre ce chapitre, vous êtes supposés connaître les bases du fonctionnement d'Internet, y compris le concepts des **sockets**, les adresses **IP** et les ports. Plus d'informations sur ces concepts peut être trouvée dans Wikipedia.

Astuce : Les protocoles standards soutenus par LiveCode comme **http** et **ftp**, discutés précédemment dans ce chapitre, ont tous été mis en œuvre dans une bibliothèque scriptée avec le socket de LiveCode. Vous pouvez examiner cette bibliothèque en exécutant le script d'édition du bouton "**revlibURL**" de la pile "**revLibrary**" dans la boîte de message. Attention, cette bibliothèque n'est pas pour la galerie. Si vous changez quelque chose, les commandes Internet de LiveCode peuvent cesser de fonctionner.

Ouverture d'une connexion

Pour ouvrir une connexion, utilisez la commande **open socket**. La commande suivante ouvre une connexion pour l'adresse **IP** spécifiée dans la variable **tIPAddress** et le port spécifié dans la variable **tPort**.

Elle spécifie que LiveCode doit envoyer le message "chatConnected" lorsque la connexion a été établie.

```
open socket (tIPAddress & ":" & tPort) with message "chatConnected"
```

Pour ouvrir un socket sécurisé, utilisez la variante **open secure socket** de la commande. Pour ouvrir un socket datagramme **UDP**, utilisez la variante **open datagram socket** de la commande.

Pour plus d'informations sur ces variantes, voir le dictionnaire de LiveCode.

Recherche d'un nom d'hôte ou l'adresse IP

Vous pouvez rechercher une adresse **IP** depuis un nom d'hôte avec la fonction **hostNameToAddress**. Par exemple, pour obtenir l'adresse **IP** du serveur runrev.com :

```
put hostNameToAddress("www.runrev.com") into tIPAddress
```

Pour obtenir le nom d'hôte de la machine locale, utilisez la fonction **hostName**. Pour rechercher le nom d'une adresse **IP**, utilisez la fonction **hostAddressToName**.

La lecture et l'écriture de données

Une fois que LiveCode ouvre une connexion, il envoie un message **chatConnected**. Pour recevoir des données, utilisez la commande **read from socket**. Le message suivant lit des données à partir du **socket** et envoie un message **chatReceived** quand la lecture est terminée.

```
on chatConnected pSocket
  read from socket pSocket with message chatReceived
end chatConnected
```

Une fois que la lecture depuis le socket est terminée le message **chatReceived** peut être utilisé pour traiter ou afficher les données. Il peut ensuite spécifier qu'il continue à lire dans le socket jusqu'à ce que plus aucune données ne soient reçues, envoyant un autre message **chatReceived** une fois terminé.

```
on chatReceived pSocket, pData
  put pData after field "chat output"
  read from socket pSocket with message "chatReceived"
end chatReceived
```

Pour écrire des données sur le **socket**, utilisez la commande d'écriture :

```
write field "chat text" to socket tSocket
```

Déconnecter

Pour déconnecter, utilisez la commande **close socket**. Vous devez stocker une variable avec les détails sur tout socket ouvert, et pouvoir les fermer lorsque vous avez fini de les utiliser ou lorsque votre pile se ferme.

```
close socket (tIDAddress & ":" & tPort)
```

Écoute et acceptation des connexions entrantes,

Pour accepter des connexions entrantes sur un port donné, utilisez la commande **accept connections**. L'exemple suivant indique à LiveCode d'écouter les connexions sur le port 1987 et d'envoyer le message **chatConnected** si une connexion est établie. Vous pouvez alors commencer à lire les données du **socket** dans le gestionnaire **chatConnected**.

```
accept connections on port 1987 with message chatConnected
```

Gestion des erreurs

S'il y a une erreur, LiveCode va envoyer un message **SocketError** avec l'adresse du **socket** et le message d'erreur. Si un **socket** est fermé un message **socketClosed** sera envoyé. Si un **socket** expire en attente de données, un message **socketTimeout** sera envoyé. Pour obtenir une liste des **sockets** qui sont ouverts, utilisez la fonction **openSockets**. Vous pouvez régler le délai d'attente par défaut en définissant la propriété **socketTimeOutInterval**. Pour plus de détails sur toutes ces fonctionnalités, voir le dictionnaire de LiveCode.

Astuce : Vous pouvez voir une implémentation complète d'une application de base client serveur **chat** ici : **Barre Icônes > Ressources > Ressources Center > Sample Projects > Internet Chat.** Téléchargez la pile. La plupart des scripts pour la pile "serveur" sont dans le bouton "start server". La plupart des scripts pour le client sont dans le script de la pile pour la pile "chat client".

Chapitre 12 Extension des Fonctions Intégrées

Ce chapitre explique comment étendre les capacités intégrées de LiveCode. Ce thème est très utile pour tous ceux qui veulent étendre l'ensemble des fonctionnalités de LiveCode. Il est également utile si vous prévoyez d'utiliser LiveCode comme un front-end à n'importe quelle application existante ou ensemble de processus.

Il existe de nombreuses façons d'étendre LiveCode. Cette rubrique explique comment exécuter des commandes shell, lancer d'autres applications, lire et écrire à des processus, exécutez AppleScript, VBScript, envoyer et répondre à AppleEvents et de communiquer entre plusieurs processus basés sur LiveCode. Il vous indique également où trouver de des informations pour créer des commandes et des fonctions (code écrit dans des langages de bas niveau) externes. Nous détaillons comment étendre le LiveCode IDE aussi: comment créer un plug-in ou modifier l'IDE lui-même.

12.1 Communiquer avec d'Autres Processus et Applications

12.1.1 Lecture et écriture pour l'interpréteur de commandes (shell)

Utilisez la fonction **Shell** pour exécuter des commandes **shell** et renvoyer le résultat. L'exemple suivant affiche une liste de répertoires sous Mac OS X:

```
answer shell("ls")
```

Et cet exemple stocke une liste de répertoire dans une variable sous Windows :

```
put shell("dir") into tDirectory
```

Sur les systèmes Windows, vous pouvez empêcher l'affichage de la fenêtre de terminal à partir en définissant la propriété globale **hideConsoleWindows** sur vrai.

Vous pouvez choisir un programme **shell** différent en définissant la propriété globale **shellPath**. Par défaut, il est réglé sur **/ bin / sh** sur Mac OS X et Linux et **command.com** sur Windows.

Astuce : La fonction **shell** bloque LiveCode jusqu'à ce qu'elle soit terminée. Si vous souhaitez exécuter une commande **shell** en tâche de fond, écrire le script **shell** dans un fichier texte puis exécuter avec la commande de lancement.

12.1.2 lancer d'autres applications

Utilisez la commande **launch** pour lancer d'autres applications, documents ou **URL**. Pour lancer une application, indiquez le chemin complet de l'application. L'exemple suivant ouvre un document texte avec TextEdit sous Mac OS X :

```
launch "/Users/someuser/Desktop/text document.rtf" with "/Applications/TextEdit.app"
```

Astuce : Pour obtenir le chemin d'accès à une application, utilisez la commande **answer file** pour choisir l'application, puis copiez la variable **it** dans votre script. Exécuter ceci dans la boîte de message :

```
answer file "Select an application"; put it
```

Pour ouvrir un document avec l'application à laquelle il est associé utilisez la commande **launch document**.

```
launch document "C:/My document.pdf"
```

Pour ouvrir une **URL** dans le navigateur Web par défaut, utilisez la commande **URL** de lancement.

```
launch URL "http://www.runrev.com/"
```

Pour plus d'informations sur le lancement d'**URL**, voir le chapitre 12. Pour plus de détails sur la façon de rendre les pages web au sein LiveCode, voir la section sur **revBrowser**.

12.1.3 Fermeture d'une autre application

Utilisez la commande **kill process**, pour envoyer un signal vers une autre application, pour la refermer ou forcer à quitter. Pour plus de détails, voir le Dictionnaire du LiveCode.

12.1.4 Communiquer avec d'autres processus

Utilisez la commande **open process** pour ouvrir une application ou un processus dont vous voulez lire et écrire des données. Vous pouvez alors y lire le processus avec la commande **read from process**, et y écrire avec la commande **write to process**. Pour fermer un processus que vous avez ouvert, utilisez la commande **close process**. **openProcesses** qui renvoie une liste des processus ouverts et **openProcessIDs** qui renvoie les **ID** des processus de chacun. Pour plus de détails, voir le Dictionnaire du LiveCode.

12.1.5 Utilisation d'AppleScript et VBScript

Pour exécuter des commandes à l'aide d'**AppleScript** sur Mac OS ou VBScript sur Windows, utilisez la commande **do as**. **do as** vous permet également d'utiliser d'autres langages **Open Scripting Architecture** sur Mac OS, ou des langages installés dans **Windows Scripting Host** sur Windows. Pour récupérer la liste des langues installées disponibles, utilisez **alternateLanguages**.
Par exemple, pour exécuter un **AppleScript** qui place le **Finder** sous Mac OS X au premier plan , saisissez la commande suivante dans un champ :

```
tell application "Finder" activate
end tell
```

Ensuite, exécutez :

```
do field 1 as "appleScript"
```

Pour récupérer un résultat commandes exécutées à l'aide **do as**, utilisez la fonction **result**. Un message d'erreur sera également retourné dans le résultat.
L'exemple suivant affiche le résultat d'une addition effectuée à l'aide de **VBScript**:

```
do "result = 1 + 1" as "vbscript"
answer the result
```

Pour plus d'informations sur la commande **do as**, voir **do** dans le Dictionnaire du LiveCode.

12.1.6 AppleEvents

Pour envoyer un **AppleEvent**, utilisez la commande **send to program**.
Si LiveCode reçoit un **AppleEvent** il enverra un message **AppleEvent** à la carte actuelle. Interceptez ce message pour effectuer des actions telles que le traitement d'une demande de quitter votre application ou ouvrir un document.
L'exemple suivant montre comment vous pouvez lancer requête d'arrêt :

```
on appleEvent pClass, pID, pSender
  if pClass & pID is "aevtquit" then
-- Appelle une fonction qui demande à l'utilisateur de sauvegarder les changements
    put checkSaveChanges() into tOkToQuit
-- Retourne faux si l'utilisateur choisit "cancel"
  if tOkToQuit is true then quit
     else exit appleEvent
  end if
end appleEvent
```

Pour obtenir des renseignements supplémentaires, passés avec **AppleEvent** utilisez la commande **request appleEvent data**. L'exemple suivant montre comment vous pouvez traiter une demande d'ouverture de pile :

```
on appleEvent pClass, pID, pSender
  --appleEvent est envoyé quand une pile est ouverte depuis le finder
  if pClass & pID is " aevtodoc " then
    -- obtenir le(s) chemin(s) fichier(s)
    request AppleEvent data
    put it into tFilesList
    repeat for each line x in tFilesList
      go stack x
    end repeat
end appleEvent
```

Pour plus de détails, voir le Dictionnaire de LiveCode.

12.1.7 Utilisation des sockets locaux

Si vous souhaitez communiquer entre les applications locales, une technique courante, qui peut être utilisée sans modification de code sur toutes les plateformes que LiveCode supporte, consiste à ouvrir un **socket** local et à communiquer avec cela. Vous devez choisir un numéro de port qui n'est pas utilisé par un protocole standard - généralement un nombre élevé.

Cette technique est couramment utilisée lorsque vous voulez créer plusieurs programmes qui fonctionnent indépendamment mais en communiquant les uns avec les autres.
C'est une technique viable pour exécuter les tâches de fond et fournit un moyen simple de créer une application qui se comporte comme si elle était ***threaded*** - c'est à dire avec des prestations de plusieurs ***threads***. Vous pouvez concevoir votre application de sorte que des instances supplémentaires puissent être lancées pour effectuer le traitement, le transfert de données ou d'autres activités intensives. Les systèmes d'exploitation modernes vont allouer à chaque application à un cœur de processeur approprié. En utilisant un message socket pour communiquer avec chacun d'eux vous pouvez garder l'interface utilisateur de votre application principale réactive et afficher les informations d'état.

```
open socket to "127.0.0.1:10000" with message gotConnection
```

Une discussion détaillée sur la façon de créer un protocole utilisant les sockets se trouve dans le chapitre 12.

Astuce : Pour faciliter la communication entre les multiples programmes LiveCode, pensez à écrire une simple bibliothèque qui envoie et reçoit le nom du gestionnaire avec les données de paramètres. Pour appeler un gestionnaire depuis l'autre programme LiveCode, envoyez le nom du gestionnaire et les données de la bibliothèque. La bibliothèque va envoyer les données sur un **socket**. Dans le programme de réception interceptez les données entrantes du **socket** et utilisez-les pour appeler le message approprié avec les données de paramètres reçues.

12.2 Extension de l'IDE LiveCode

L'IDE (environnement de développement intégré) LiveCode a été écrit en utilisant LiveCode. Tous les composants - la palette d'outils, l'inspecteur de propriété, l'éditeur de scripts, le débogueur, etc... sont mis en œuvre comme des piles de LiveCode. L'IDE a une série de bibliothèques **frontScripts** et **backScripts**, qu'il utilise pour fournir des fonctionnalités à la fois pour l'IDE et à votre application.

Certaines de ces bibliothèques ne sont utilisées que par l'IDE (par exemple, la bibliothèque débogueur), d'autres (par exemple, la bibliothèque de l'Internet, **libURL**) sont copiées dans votre application ***standalone*** par le constructeur d'application.

Cette conception permet facilement d'étendre l'IDE avec des plug-ins. Si vous êtes un développeur de LiveCode avancé, vous pouvez également modifier l'IDE lui-même pour offrir des fonctionnalités personnalisées.

12.2.1 Création de Plug-ins

Vous pouvez créer un plug-in pour aider à accomplir les tâches que vous devez faire régulièrement dans l'IDE de LiveCode. Les plug-ins sont écrits comme des piles de LiveCode. (Si vous avez besoin d'étendre LiveCode utilisant un langage de niveau inférieur, voir la section sur Externals, ci-dessous.)

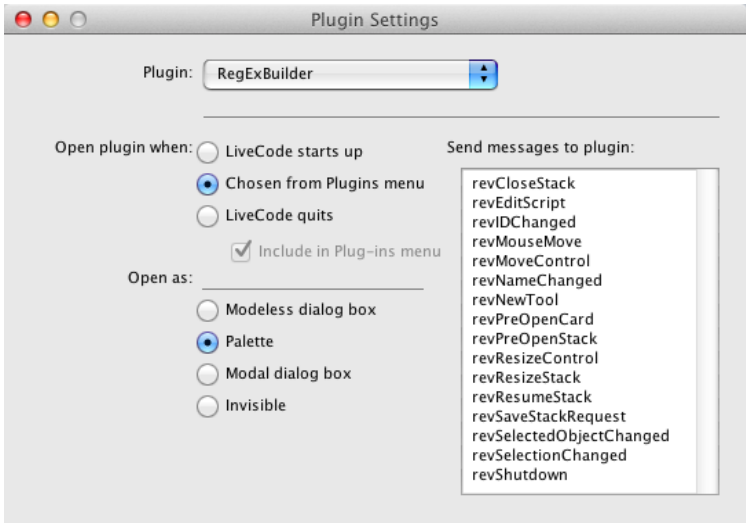
Pour créer un plug-in, enregistrez votre pile dans le dossier **Plugins**, situé dans le dossier **My LiveCode** [Edition] à l'intérieur de votre dossier **Documents**.

Vous pouvez maintenant charger votre pile en choisissant son nom dans le menu Développement > sous-menu **Plugins**. Par défaut, votre plug-in sera chargé comme une palette. Cela vous permet d'utiliser les commandes dans le plugin tandis que l' **IDE** LiveCode est en mode outil pointeur. Cela vous permet de créer des comportements personnalisés de style **PropertyInspector**, ou d'autres outils d'édition d'objet.

L'écran Paramètres du Plugin

Ouvrez l'écran des paramètres du **plug-in** : **menu developpement > sous-menu Plugins**. Choisissez le plugin que vous avez créé à partir du menu du plugin en haut de l'écran pour lui appliquer les paramètres.

Open plug-in when : Par défaut, votre **plugin** va se charger lorsque vous l'avez choisi dans le menu **plugins**. Si vous voulez avoir votre **plugin** chargé, chaque fois que vous démarrez LiveCode sélectionnez l'option



LiveCode starts up : Utilisez cette option si votre plugin est utilisé pour configurer votre environnement, par exemple par des piles en cours d'utilisation ou pour ajuster la disposition de la fenêtre.
Pour que votre **plugin** soit chargé lorsque LiveCode est en train de quitter, choisir **LiveCode quits**. Utilisez cette option si votre **plugin** exécute un nettoyage des tâches que vous souhaitez voir exécuté chaque fois que vous quittez.

Ouvrir en tant que : Choisissez le mode voulu pour lancer votre **plugin** en tant que.
Si vous choisissez l'option **invisible**, votre pile d'extension sera chargée en restant invisible. Utilisez cette option pour créer un **plugin** qui installe une bibliothèque anonyme ou pour effectuer une autre tâche automatisée qui ne nécessite pas une interface utilisateur visible.

Remarque : Le chargement de votre **plugin** ne vous permettra pas de modifier le **plugin** lui-même. Si vous souhaitez modifier le **plugin**, d'abord le charger dans le menu puis utilisez le navigateur de l'application pour le monter au premier niveau par un clic droit dessus dans la liste des piles et en choisissant **Toplevel** dans le menu contextuel.

Envoyer des messages aux plugin : Pour avoir votre **plugin** réactif pendant que vous travaillez dans l'**IDE** vous devez l'enregistrer pour recevoir des messages.
L'**IDE** peut envoyer une série de messages à la carte en cours dans votre **plug-in** que vous changiez de sélection, d'outils de commutation, que les piles s'ouvrent et se ferment, etc.

Les messages qui peuvent être envoyés sont énumérés ci-dessous.

Message	Envoyé quand
RevCloseStack	L'utilisateur ferme une pile dans l'IDE
revEditScript	L'utilisateur choisit l'option "Modifier le script"
revIDChanged	L'identifiant d'un objet est modifié
revMouseMove	La souris est déplacée
revMoveControl	Un contrôle est déplacé avec l'outil pointeur
revNameChanged	Le nom d'un objet est modifié
revNewTool	Un nouvel outil est choisi
revPreOpenCard	Un message est envoyé sur preOpenCard au changement de carte
revPreOpenStack	Un message de preOpenStack est envoyé sur l'ouverture pile
revResizeControl	Un contrôle est redimensionné avec l'outil pointeur
revResizeStack	Une pile est redimensionnée
revResumeStack	Une pile est activée
revSaveStackRequest	La commande de sauvegarde est exécutée
revSelectedObjectChanged	La sélection est modifiée avec l'outil pointeur
revSelectionChanged	La sélection de texte est modifiée
revShutdown	LiveCode est quitté

Par exemple, pour avoir votre plugin à jour chaque fois qu'un objet est sélectionné avec l'outil pointeur, sélectionnez le message **revSelectedObjectChanged**.

Ensuite, insérer le gestionnaire suivant dans le script de la carte du plugin :

```
on revSelectedObjectChanged
  -- stocke la liste des objets sélectionnés
  put the selObj into tObjectsList
  repeat for each line l in tObjectsList
    --insérer le code à utiliser sur chaque objet ici
  end repeat
end revSelectedObjectChanged
```

12.2.2 Modification de l'IDE

Attention : Modifier l'IDE peut facilement rendre LiveCode inutilisable. Nous recommandons que seuls les utilisateurs avancés tentent de modifier l'IDE. Nous vous conseillons de sauvegarder l'IDE avant de faire des changements. Nous vous déconseillons de tenter de modifier l'IDE tout en travaillant sur un projet critique.

Pour modifier l'**IDE** LiveCode, activez **Live Code UI Elements in Lists** depuis le menu Affichage. Cela active LiveCode pour afficher ses propres piles dans le navigateur de l'application et les autres écrans d'édition. Vous pouvez maintenant charger ces piles pour les éditer. Pour vous permettre de modifier des objets de l'**IDE** LiveCode avec les raccourcis clavier, activer les options **In Live Code UI Windows** et **Contextual menus work in LiveCode Windows** dans les préférences.

Attention : Si vous faites une erreur d' édition de **revFrontScript** ou **revBackScript**, LiveCode deviendra insensible et vous devrez forcer son arrêt

L'**IDE** utilise la pile **revLibrary**, afin de fournir une grande partie de ses fonctionnalités. Les scripts sont stockés sous forme d'une série de boutons et chargés dans les **frontScripts** et **backScripts** lorsque l'**IDE** est démarré. Pour modifier ces scripts, allez à l'onglet **Back Scripts** ou **Front Scripts** dans la boîte de message et cochez la case **Show Live Code UI Scripts**.

12.3 Externals - code écrit dans des langages de bas niveau

LiveCode fournit une interface externe qui vous permet de l'étendre à l'aide d'un langage de niveau inférieur (souvent C). Par exemple, si vous avez du code qui effectue un traitement dans un langage de bas niveau préexistant, vous pouvez écrire une interface utilisateur depuis LiveCode et ensuite appeler cette bibliothèque en écrivant un simple **wrapper** autour en utilisant l'interface **externals** de LiveCode. LiveCode supporte la transmission de données vers et depuis les codes externes, le dessin dans les objets image dans des fenêtres LiveCode, la manipulation de l'objet **player**, et plus encore.

Remarque : Certains aspects de la fonctionnalité de commande interne sont fournis sous la forme de codes externes. Il s'agit notamment de la bibliothèque **SSL**, de la bibliothèque de base de données, de la bibliothèque **revBrowser**, de la bibliothèque **zip**, de la carte d'acquisition vidéo et des bibliothèques **XML**. Ces bibliothèques peuvent être incluses dans une application autonome, ou exclues dans le cas contraire - pour économiser de l'espace disque.

12.3.1 Externals

http://livecode.com/developers/guides/desktop-externals/

12.4 Construction d'une application Web

Des documents supplémentaires pour cette section sont en préparation pour la prochaine version de

LiveCode. En attendant, nous vous recommandons le tutoriel complet sur ce sujet ici :
http://www.hyperactivesw.com/cgitutorial/intro.html

Chapitre 13 Travailler Avec Les Médias

L'une des utilisations les plus populaires de LiveCode consiste à créer de véritables applications multimédia. Même si vous n'êtes pas en train de créer une application multimédia classique, de nombreuses applications nécessitent une interface utilisateur attrayante. Ce chapitre détaille le support média de LiveCode.
Nous examinons les caractéristiques et capacités des images : comment importer et exporter des images dans une variété de formats, comment manipuler des images à l'intérieur LiveCode, travailler avec des masques, le presse-papiers et la capture d'écran, en passant par des GIF animés.
Nous détaillons les caractéristiques des graphiques vectoriels et expliquons comment les manipuler par script. Nous traitons ensuite de la vidéo et des fonctionnalités audio.
Ensuite, nous vous montrons comment créer des boutons personnalisés avec un thème. Finalement nous donnons un aperçu du support pour des effets de transition visuels.

13.1 Images Bitmap

LiveCode supporte une grande variété de formats d'image, y compris les formats **JPEG** et **PNG** populaires. Les images **PNG** peu encombrantes supportent totalement les canaux alpha., les images **JPEG** pour afficher des photos. Pour plus de détails, voir les tableau ci-dessous.

Format	Export	Masque	Commentaires
PNG	OUI	OUI, 1-bit ou canal alpha	Prend en charge le réglage de gamma, prend en charge les canaux alpha, l'entrelacement
JPEG	OUI	NON	Fichiers JPEG progressifs; compression avec perte. Export vous permet de définir le niveau de compression
GIF	OUI	1-bit	GIF87a GIF89a ; supporte l'animation ; supporte les GIF entrelacés ; maximum de 256 couleurs
BMP	NON	NON	Non compressé
PBM	OUI	NON	1 bit (noir et blanc)
PGM	NON	NON	Niveaux de gris
PPM	NON	NON	
XBM	NON	NON	
XPM	NON	NON	1 bit (noir et blanc)
XWD	NON	NON	
PICT	NON	NON	non compressé

Comme vous pouvez le voir dans le tableau ci-dessus, un certain nombre de formats d'image pris en charge peuvent également être exportés par LiveCode.
Vous pouvez modifier des images en utilisant les outils de dessin de LiveCode, ou manipuler les données binaires par script ou en utilisant un fichier externe.
Vous pouvez créer des images à partir de l'un des objets natifs de LiveCode, y compris des boutons, des champs et des graphiques. Ceux-ci peuvent ensuite été exportés dans de nombreux formats. Vous pouvez copier des images à partir du presse-papiers ou les exporter vers le presse-papiers.
LiveCode peut capturer une partie de l'écran, ou la totalité de l'écran.

13.1.1 Importer des Images

Pour importer une image, choisissez **File** > **Import As Control** > **Image File**. Sélectionnez le fichier image que vous souhaitez importer. Cela va importer le fichier image dans un nouvel objet image sur la carte actuelle.
Cela équivaut à exécuter la commande **import paint** dans la boîte de message.

Note : Si vous souhaitez réutiliser une image dans votre application, par exemple dans le cadre d'un habillage personnalisé pour votre application, créez une sous-pile et importez-y toutes vos images. Vous pouvez ensuite les référencer sur l'ensemble de votre fichier de pile. Pour plus d'informations, consultez la section Création de Skins personnalisés, ci-dessous.

13.1.2 Importer des Images Référencées

Pour référencer un fichier image sur le disque choisir **File** > **New Referenced Control** > **Image File**. Cela crée un objet image sur la carte actuelle et définit sa propriété **fileName** avec le chemin du fichier image que vous avez sélectionné. Les images référencées n'augmentent pas la taille de votre pile, et ne sont chargées qu' à partir du disque en mémoire lorsque vous accédez à la carte courante ou, à défaut, vous les affichez sur l'écran. Les images référencées sont idéales lorsque vous souhaitez mettre à jour l'image dans un éditeur d'image et voir les changements dans LiveCode sans avoir besoin de la ré-importer.

Vous pouvez envisager la création d'un dossier à côté de votre fichier de pile pour contenir les fichiers image puis à l'aide de l'inspecteur de modifier le chemin d'accès à l'image pour être un chemin référencé. Cela vous permet de déplacer la pile et le dossier des images ensemble sur un système différent. Pour plus d'informations sur les chemins de fichiers référencés, voir la section Nom de fichier Spécifications et chemins.

Vous pouvez utiliser le constructeur d'applications pour copier les images référencées dans un répertoire (mise à jour de la propriété **fileName** de chaque image) ou pour copier les images référencées dans votre pile. Pour plus d'informations, voir le chapitre Déploiement de Votre Application.

Important : Vous ne pouvez pas utiliser les outils de peinture ou manipuler les données binaires d'images référencées. Vous devez les importer en premier. Si vous voulez modifier le fichier original, vous pouvez apporter des modifications puis exporter l'image - voir ci-dessous pour plus de détails.

13.1.3 Importation en utilisant Screen Capture

Pour importer une image en captant une partie de l'écran, choisissez **File** > **Import As Control** > **Snapshot**. Sélectionnez la zone de l'écran que vous souhaitez importer.
Pour prendre une capture d'écran par script, utilisez la commande **import snapshot**. Pour spécifier une section de l'écran pour importer sans afficher le collimateur utilisez **import snapshot from rect** :

```
import snapshot from 100,100,200,200
```

Cela va créer un objet image sur la carte actuelle à partir de la zone rectangulaire spécifiée.

13.1.4 Création d'images

Pour créer une image, faites glisser un objet image à partir de la palette d'outils vers votre pile. Vous pouvez maintenant peindre sur l'image en utilisant les outils de dessin, définir la référence **fileName** d'une image ou manipuler les données binaires de l'image.

13.1.5 Utilisation des outils de peinture

Pour accéder aux outils de dessin, appuyez sur le triangle, en bas à droite de la palette Outils.

	Outil	Usage	Raccourcis Clavier
	Sélecteur	Faites glisser pour sélectionner une zone rectangulaire d'une image	**Maj** force un carré **cmd / ctrl** sélection des doubles
	Seau	Remplir les formes avec la couleur. Remplira tout pixel connecté au pixel cliqué avec la couleur du pinceau	**Ctrl / clic** remplir avec transparence
	aérosol	Dessiner une couleur peinte avec l'aérosol en utilisant la forme du pinceau	**Ctrl / clic** diffuser avec transparence
	Gomme	Enlever la couleur d'une zone en la laissant transparente. Utilise la forme du pinceau	
	Rectangle	Dessiner une forme de rectangle	**Maj** force un carré **Ctrl** crée de la transparence
	Rectangle arrondi	Dessiner une forme de rectangle arrondi (maintenez sur le rectangle pour sélectionner une forme)	**Maj** force un carré **Ctrl** crée de la transparence
	Ovale	Dessiner une forme ovale (maintenir sur le rectangle pour sélectionner une forme)	**Maj** force un carré **Ctrl** crée de la transparence
	Polygone régulier	Dessiner la forme de polygone régulier (maintenir sur le rectangle pour sélectionner une forme)	**Maj** force un carré, force rotation, **Ctrl** crée de la transparence
	Polygone	Dessiner la forme du polygone (maintenir sur le rectangle pour sélectionner une forme)	**Maj** force lignes avec angles multiples de 22,5 ° **Ctrl** crée de la transparence
	Ligne	Tracer une ligne droite	**Maj** force lignes avec angles multiples de 22,5 ° **Ctrl** crée de la transparence
	Main levée	Dessiner une courbe à main levée (maintenir sur la forme de la ligne pour sélectionner une forme). Si l'option de remplissage est choisie le début et la fin de la courbe sont jointes automatiquement lorsque vous avez fini de dessiner	**Alt / option** empêche de dessiner sur la ligne de bordure **Ctrl** crée de la transparence
	Crayon	Tracez une ligne à main levée d'un pixel de largeur	**Ctrl** crée de la transparence
	Pinceau	Dessiner des traits au pinceau en utilisant la forme du pinceau	**Ctrl** crée de la transparence **Cmd / clic** pour agrandir
	Remplissage (pinceau) couleur	Choisissez une couleur pour remplir des formes ou utilisez les outils de pinceau	
	Couleur Ligne	Choisissez une couleur pour dessiner des lignes ou utilisez avec l'outil Crayon	
	Forme de la brosse	Choisissez une forme de brosse pour utiliser avec les outils : pinceau, gomme, aérographe	

Pour agrandir une image, cliquez droit dessus avec l'outil de pointeur et choisissez **Magnify** dans le menu. Lorsque vous modifiez une image, elle sera recompressée dans le format spécifié par la propriété globale **paintCompression**.

13.1.6 Scripts avec les outils de peinture

Dessiner par script, peut être utile si vous souhaitez que l'utilisateur puisse voir chaque action de dessin. Si vous souhaitez manipuler les données d'une image plus efficacement hors de l'écran, voir la section suivante. Afin de maîtriser les outils de dessin par script, créer une image, puis choisir l'outil de dessin que vous

souhaitez utiliser. Réglez la brosse appropriée, le motif ou de taille de ligne puis utilisez la commande **drag** pour dessiner.
L'exemple suivant crée une image, choisit l'outil pinceau, sélectionne une petite forme de la brosse circulaire, choisit une couleur rouge, puis trace une ligne :

```
set the rect of the templateImage to 100,100,400,400 -- règle la taille de l'image
create image
choose brush tool
set the brush to 8
set the brushColor to red -- on peut utiliser les valeurs RGB
set the dragSpeed to 20 -- vraiment lent
drag from 120,120 to 300,300
```

Pour plus d'informations, consultez les entrées pour **templateImage**, **tool**, **brush**, **brushColor**, **brushPattern**, **dragSpeed, penColor** et **penPattern** dans le Dictionnaire du LiveCode.

Vous pouvez réduire la taille d'une image en utilisant la commande **crop**. Vous pouvez faire pivoter une image en utilisant la commande **rotate**.

Pour régler la qualité de l'algorithme de mise à l'échelle utilisée lors du redimensionnement d'une image, définissez la propriété **resizeQuality** avant de régler le **rect** de l'image.

13.1.7 Manipulation des données binaires d'image

Pour manipuler les données binaires d'une image, utilisez la propriété **imageData** de l'image. Cette propriété renvoie couleur, et la valeur de la transparence de chaque pixel de l'image dans un format compatible quel que soit le format auquel elle est enregistrée. **imageData** est stocké sous forme binaire, chaque pixel représenté par 4 octets. Pour convertir vers et à partir des valeurs **RVB**, utiliser les fonctions **charToNum** et **numToChar**.

Par exemple, la valeur numérique du canal rouge du dixième pixel est donnée par l'expression :

```
charToNum(char (4 * 9) + 2 of the imageData of image)
```

La valeur numérique du canal vert est :

```
charToNum(char (4 * 9) + 3 of the imageData of image)
```

et la valeur numérique de la voie bleue est :

```
charToNum(char (4 * 9) + 4 of the imageData of image)
```

Pour manipuler les données binaires d'une image en utilisant un externe, utilisez la propriété **imagePixMapID**. Lorsque vous réglez **imageData** sur l'image, l'image sera recompressée dans le format spécifié par la propriété globale **paintCompression**.

13.1.8 rendu d'une image à partir d'objets

Utilisez la commande **import snapshot** pour créer une image à partir d'objets ou d'une zone d'une pile. Au lieu de spécifier un rectangle en coordonnées globales (comme décrit ci-dessus) spécifier une pile ou un objet.

Pour importer un instantané d'une zone d'une pile :

```
import snapshot from 100,100,200,200 of stack "Layout"
```

Note : Contrairement à la capture d'écran, la pile ou l'objet que vous indiquez pour importer une image à partir de, n'a pas besoin d'être affichée à l'écran. Vous pouvez créer une mise hors écran dans une pile invisible puis faire une restitution dans une image.

Pour importer un instantané d'un objet :

```
import snapshot from button 5 of stack "objects"
```

La commande importsnapshot crée une nouvelle image dans le **defaultStack** actuel. L'image est codée en utilisant le format de **paintCompression** actuel.

Pour enregistrer cet instantané directement dans un fichier au lieu de créer une image, utilisez la commande **export snapshot** :

```
export snapshot from the selectedObject to file "snap.jpg" as JPEG
```

13.1.9 Exportation d'images

Pour exporter une image dans le format actuel, dans lequel elle est stockée, la mettre dans un fichier binaire en utilisant les commandes **URL**. L'exemple suivant invite l'utilisateur à sélectionner un fichier, puis exporter l'image en lui :

```
ask file "Select a file:"
put image "picture" into URL ("binfile:" & it)
```

Pour exporter une image dans un format différent, utilisez la commande **export**.

```
export image "picture" to file "picture.png" as PNG
```

Vous pouvez également exporter une image à une variable. Voir la commande d'exportation dans le Dictionnaire du LiveCode pour plus d'informations.

13.1.10 Copier et coller des images

Pour copier une image en interne sans utiliser le presse-papiers du système, il suffit de la mettre dans une variable ou dans une autre image.

```
put image 1 into image 2
```

Pour recompresser l'image dans un format différent, utilisez la commande **export** pour l'exporter vers une variable ensuite mettre cette variable dans une image.

Pour copier une image dans le presse-papiers, utilisez la commande **copy**.

```
copy image 1
```

Pour coller une image à partir du presse-papiers, utilisez la commande **paste**.

Pour transférer les données binaires sur l'image vers et depuis le presse-papiers, obtenir et définir la propriété **clipboardData ["image"]**.
Voir l'entrée de **clipboardData** dans le Dictionnaire du LiveCode pour plus d'informations.

13.1.11 Travailler avec des GIF animés

Vous pouvez importer une image **GIF** animée de la même façon que d'autres images.

Définissez la propriété **repeatCount** pour spécifier la fréquence de jeu de l'animation.
Le réglage de **repeatCount** à 0 arrête l'animation et à -1 l'amène à se répéter éternellement. Pour changer le cadre actuel, définissez la propriété **currentFrame**.

*Note : Si vous utilisez un **GIF** animé comme une icône de bouton, il s'animera simultanément dans chaque bouton dans lequel il est utilisé.*

13.2 Travailler avec des graphiques vectoriels

En plus des images bitmap, LiveCode prend également en charge les graphiques vectoriels. Vous pouvez créer des graphiques vectoriels en utilisant les outils graphiques, ou par script. Vous pouvez les manipuler de façon interactive dans l'**IDE** ou par script. Vous pouvez modifier les couches, exporter leur descripteurs ou les convertir au format bitmap.

Astuce : Animation Engine est une bibliothèque de tierce partie qui comprend un ensemble de fonctions pour la programmation des animations graphiques interactives. Voir la section téléchargements associés, de notre site web pour plus d'informations.

13.2.1 Outils Graphiques Vectoriels

Pour voir les outils graphiques, déplier le triangle au bas à droite de la palette d'outils. Les outils graphiques sont situés au-dessus des outils de peinture sur la palette d'outils. Les outils graphiques fonctionnent de la même manière que les outils de dessin, sauf que chaque fois que vous dessinez une forme un nouvel objet graphique est créé. Contrairement graphismes de peinture, vous pouvez redimensionner et ajuster des objets graphiques une fois que vous les avez créés. Pour plus d'informations sur les formes individuelles, voir la section sur les outils de peinture, ci-dessus.

13.2.2 Créer des graphiques par script

Pour créer des graphiques par script, définissez les propriétés de **templateGraphic** puis utilisez la commande **create graphic**. Pour plus d'informations sur les objets modèles, consultez la section Création d'objets hors écran à l'aide des objets modèles dans le chapitre 7.

13.2.3 Manipulation des graphismes par le script

Parce que chaque graphique est un objet individuel, vous manipulez son apparence en définissant des propriétés plutôt que d'utiliser les commandes de glissé (comme avec outils de peinture, ci-dessus). Vous pouvez contrôler toutes les propriétés de l'objet graphique par un script avec les propriétés **rectangle**, **line** et **fill**.
Vous pouvez modifier un graphique d'un type à un autre (par exemple un rectangle à un ovale) en définissant sa propriété **style**.
Le style de graphique polygone possède une propriété des points qui vous permet de définir les différents points d'une ligne.
Un mouvement simple peut être appliqué en utilisant la commande **move**. Par exemple, pour déplacer un graphique 100 pixels vers la droite de manière asynchrone :

```
move graphic 1 relative 100,0 without waiting
```

Pour plus d'informations, voir la commande de déplacement dans le Dictionnaire du LiveCode.
Pour plus d'informations, voir la commande **xmove** dans le Dictionnaire du LiveCode.
Pour programmer un effet d'animation plus complexe, calculer les modifications des points ou des rectangles

et mettre ces valeurs en utilisant le timing basé sur les messages.
L'exemple suivant redimensionne un graphique nommé "rectangle" par 100 pixels en 1 seconde.

```
local lCount
on mouseUp
  put 0 into scaleGraphic
  scaleGraphic
end mouseUp
on scaleGraphic
  add 1 to lCount
  if lCount > 100 then exit scaleGraphic
  get the rect of graphic "rectangle"
  add 1 to item 1 of it
  add 1 to item 2 of it
  subtract 1 from item 3 of it
  subtract 1 from item 4 of it
  set the rect of graphic "rectangle" to it
  send "scaleGraphic" to me in 10 milliseconds
end scaleGraphic
```

13.3 Utilisation de la vidéo

LiveCode charge la lecture de vidéo avec l'objet lecteur. Sous Windows et Mac OS, c'est QuickTime qui est pris en charge. QuickTime permet la lecture d'une grande variété de formats de fichiers, y compris **MPEG**, **H.264** et **AAC**. LiveCode prend également en charge le contrôleur, accédant aux fonctionnalités de streaming de QuickTime et QuickTime Virtual Reality (**QTVR**).

Sur les systèmes Windows, l'objet lecteur peut lire une vidéo en utilisant le sous-système Windows Media. Cela est plus limité fonctionnalités qu'avec QuickTime.

Sur les systèmes Linux, l'objet lecteur peut lire des vidéos en utilisant **mplayer**. Il y a quelques limitations de fonctionnalités : les propriétés **alwaysBuffer**, **startTime**, **endTime** et **playSelection** n'ont aucun effet, **Showcontroller** n'a aucun effet et avancer au pas en avant / en arrière ne fonctionne pas de manière fiable.

En plus de ces fonctionnalités, LiveCode a un support intégré pour le format GIF animé. Les fichiers GIF animés peuvent être lus sans logiciel de tierce partie. Voir ci-dessus pour plus d'informations. D'autres formats pris en charge par les plug-ins dans les navigateurs Web peuvent être lus en utilisant **revBrowser** (Flash par exemple). Voir le sujet sur **revBrowser** pour plus d'informations.

13.3.1 L'Objet Player (Lecteur)

Utilisez l'objet lecteur pour lire et interagir avec la vidéo et l'audio. Pour créer un objet lecteur, faites-le glisser sur votre pile depuis la palette Outils. Pour sélectionner un fichier vidéo à lire, ouvrez l'inspecteur de l'objet lecteur et sélectionnez le fichier à utiliser comme source. Faire cela définit la propriété **fileName** du lecteur.

Pour diffuser un film à partir d'un serveur Internet, définissez la propriété **fileName** à l'adresse **URL** du flux.

Pour utiliser un lecteur, utilisez les commandes **start** et **stop**.

```
start player 1
```

```
stop player 1
```

Le tableau suivant décrit les propriétés du player couramment utilisé :

Propriété	Fonction	Exemple
alwaysBuffer	Oblige le lecteur à être en mémoire tampon, permettant aux objets d'être dessinés au premier plan et au cadre courant d'être imprimé	**set** the alwaysBuffer of player 1 to true
showController	Afficher ou de masquer le contrôleur QuickTime	**set** the showController of player 1 to false
currentTime	Définit la trame actuelle	**set** the currentTime of player 1 to 1000
Duration & timeScale	La durée du film et le nombre d'intervalles par seconde d'un film	**put** (the duration of me/the timeScale of me) into tRunTime
currentTimeChanged	Un message envoyé lors de changements des trames actuelles	**on** currentTimeChanged pInterval **put** pInterval into field "Time Code" **end** currentTimeChanged
startTime	Le temps de départ de la sélection	**set** the startTime of player 1 to 500
endTime	Le temps de fin de la sélection	**set** the endTime of player 1 to 1000
showSelection	Afficher la sélection dans le contrôleur	**set** the showSelection of player 1 to true
playSelection	Lire uniquement la sélection	**set** the playSelection of player 1 to true
playRate	La vitesse de lecture de la vidéo. Réglez cette valeur à -1 pour jouer à l'envers	**set** the playRate of player 1 to 2
looping	Provoque une lecture en boucle	**set** the looping of player 1 to true
playLoudness	Régler le volume	**set** the playLoudNess of player 1 to 50
tracks	Liste des pistes dans le film	**put** the tracks of player 1 into tTracksList
enabledTracks	Activer ou désactiver les pistes	**Set** the enabledTracks of player 1 to 3
callbacks	Une liste des messages à envoyer lorsque le film atteint des moments spécifiés	**set** the callbacks of player 1 to "1000, nextScene"

Les propriétés suivantes peuvent être utilisées pour commander un film **QTVR** : **pan**, **tilt**, **zoom**, **currentNode**, **nodeChanged**, **hotspots**, et **hotSpotClicked**. Pour plus d'informations sur l'un de ces termes, voir le Dictionnaire du LiveCode.

13.3.2 Eviter l'utilisation de QuickTime sur Windows

Pour forcer LiveCode à utiliser le sous-système Windows Media au lieu de QuickTime sur les systèmes Windows, même si QuickTime est installé, définissez la propriété globale **dontUseQT** sur vrai.

Important : Vous devez configurer **dontUseQT** sur vrai avant d'exécuter n'importe quel film ou action relatifs au lecteur.

13.4 Travailler avec les sons

En plus de la lecture de sons dans une grande variété de formats en utilisant l'objet lecteur, LiveCode a un support intégré pour la lecture de clips audio au format WAV, AIFF et AU.

Note : Nous vous recommandons d'utiliser l'objet **player** pour la lecture audio, car il supporte une large gamme de formats et de types de compression. Utilisez **audioClip** lorsque vous avez besoin d'une lecture audio de base sur les systèmes qui ne disposent pas de l'une des bibliothèques de tierce partie prises en charge par l'objet **Player** installé.

13.4.1 Importation d'un clip audio

Pour importer un clip audio choisissez **File > Import as Control > Audio File**. Cela va importer le fichier audio sélectionné dans la pile actuelle comme **audioClip**.

13.4.2 Lire Un Audio Clip

Pour lire a un audio clip:

```
play audioClip 1
```

Pour arrêter :

```
play stop
```

Pour régler l volume :

```
set the playLoudness of audioclip 1 to 50 -- augmenter le volume à 50%
```

13.5 Travailler avec des effets de transition visuels

LiveCode prend en charge effets de transitions visuelles lors du changement de carte ou de masquage et l'affichage des objets.
Il existe trois types d'effet : des effets intégrés qui fonctionnent sur toutes les plateformes, des effets QuickTime qui travaillent sur des systèmes qui ont installé QuickTime et **Core Image effects** qui fonctionne depuis Mac OS 10.4 aux versions ultérieures.

Utilisez la commande **visual effect** pour afficher un effet visuel.
Pour passer à la prochaine carte avec un effet de fondu enchaîné :

```
visual effect dissolve slow
go next card
```

Pour apporter des modifications à des objets sur l'écran avec un effet visuel (par exemple, masquer, afficher ou les déplacer), d'abord verrouiller l'écran, puis effectuez les modifications, puis déverrouiller l'écran :

```
lock screen
hide image 1
show image 3
unlock screen with visual effect "wipe right"
```

Pour choisir un effet QuickTime, en utilisant la boîte de dialogue pour sélectionner l'effet, utilisez :

```
answer effect -- stocker l'effet visuel comme une proprité personnelle
set the cEffect of this stack to it
visual effect (the cEffect of this stack)
go next card
```

Pour plus d'informations sur les effets visuels, voir la commande **visual effect** dans le Dictionnaire du LiveCode.

13.6 Création Skins personnalisés

En plus de son support des contrôles natifs du système, LiveCode vous offre la possibilité de créer un look totalement personnalisé, ou habillage (**skin**), pour votre application. Tous les éléments intégrés peuvent être remplacés par des visuels sur le thème qui vous permet de créer du multimédia riche. Par exemple, les

boutons de LiveCode accepte des icônes de taille illimitée qui peuvent être utilisées pour remplacer totalement l'apparence native d'un bouton. Les fenêtres peuvent être de forme irrégulières et même contenir des manques (trous) et un masque alpha (variable) pour la transparence. Tous les objets de LiveCode supportent une variété de modes de transfert, ou d'encres. Les piles peuvent prendre en charge la totalité de l'écran ou être affichées avec un un fond.

13.6.1 Boutons personnalisés

Pour créer un bouton personnalisé, commencez par créer un bouton normal. Ouvrez l'inspecteur du bouton et mettez son style transparent. Désactivez la propriété **Show Name**. Ensuite, basculer sur le volet **Icons & Border** dans l'Inspecteur. Désactivez la propriété **HiliteBorder**.
Vous pouvez maintenant sélectionner n'importe quelle image importée pour l'utiliser comme icône pour le bouton. Pour configurer l'état de souris cliquée, définir l'icône **hilite**. Pour régler l'état de survol du curseur définir l'icône **hover**.

Important : Une icône de bouton peut être une image de n'importe quelle taille dans LiveCode, permettant un look totalement personnalisé.

Astuce : Tout objet en LiveCode peut se comporter comme un bouton. Par exemple, si vous voulez créer un graphique qui répond quand un utilisateur clique dessus, créez l'image et ajoutez-y un gestionnaire mouseUp, de la même façon que vous le feriez avec un bouton.

Astuce : Pour utiliser le même ensemble d'images sous forme d'icônes dans votre dossier de pile, créer une sous-pile et importer toutes les images de votre thème en son sein. Définissez la propriété **ID** de chaque image à une valeur élevée (entre 10.000 et 100.000), de sorte qu'il sera unique. Vous pouvez maintenant référencer cette image n'importe où dans votre application. Si vous souhaitez changer de thème, il vous suffit de remplacer cette image par une autre image et lui donner le même **ID**.

13.6.2 Fenêtres de Style Irrégulier

Pour créer une fenêtre de forme irrégulière, importer ou créez une image qui possède un masque de transparence. Ensuite, utilisez l'Inspecteur de Pile pour choisir cette image comme forme de la pile. Pour modifier la forme par script, définissez la propriété **windowShape**. Beaucoup de gestionnaires de fenêtres modernes supportent les fenêtres fusionnées avec un canal alpha (degrés variables de transparence). Pour créer une fenêtre avec un canal alpha, importer un fichier **PNG** qui contient un canal alpha et définir le **windowShape** sur cette image.

13.6.3 Modes de fusion (modes de transfert ou des encres)

Les modes de fusion déterminent la façon dont les couleurs d'un objet se combinent avec les couleurs des pixels sous l'objet pour déterminer la couleur de l'objet est affiché. Pour définir le mode de fusion d'un objet, utilisez le volet **Blending** dans l'Inspecteur ou définir la propriété **ink** de l'objet. Tous les objets, dans LiveCode, supportent les modes de fusion, à l'exception des piles.

```
set the ink of image "picture" to "blendBurn"
```

Pour plus d'informations, consultez l'entrée de **ink** dans le Dictionnaire du LiveCode.

Pour définir le degré de transparence, définissez la propriété **blendLevel** de l'objet. Tous les objets de LiveCode (y compris les piles) supportent **blendLevel** :

```
set the blendLevel of button 1 to 50 -- met le button 50% transparent
```

13.6.4 Mode plein écran

Une pile peut être affichée en plein écran en définissant sa propriété **fullScreen** sur vrai :

```
set the fullScreen of this stack to true
```

Réglez cette propriété à false pour quitter le mode plein écran.

Si vous souhaitez masquer ou afficher la barre de menu Mac OS X utiliser le **hide menubar** ou commandes **show menubar** :

```
hide menuBar
show menuBar
```

De même, utiliser **hide taskbar** et **show taskbar** sur les systèmes Windows pour afficher et masquer la barre des tâches.

13.6.5 Affichage d'une toile de fond

Pour afficher un fond définir la propriété globale **backDrop**. Vous pouvez définir la toile de fond d'une couleur unie ou à l'ID d'une image.

```
set the backDrop to "black"
```

Pour supprimer **backDrop** :

```
set the backDrop to none
```