# BeeBase

Une base de données relationnelle programmable
Version 1.1

28 septembre 2024

**Steffen Gutmann**

Note du traducteur : cette traduction française correspond à la version 1.1 de BeeBase mais certaines parties modifiées depuis la version 2.8 sont encore en anglais. Veuillez nous en excuser, nous faisons le maximum pour la compléter dans les meilleurs délais.

Traduction française d'Alexandre Balaban, Lionel Müller, Gilles Mathevet, Philippe Ferrucci, Pascal Marcelin et Stéphane Aulery 2007-2024

# Table des matières

# 1 Conditions de distribution



BeeBase est un logiciel libre distribué sous les termes de la Licence Générale Publique GNU (GPL).

## 1.1 Licence



## 1.2 Faire un don

BeeBase est complètement gratuit. Cependant, si vous aimez le projet et voulez le supporter, une donation sera toujours la bienvenue. Pour plus d'informations sur la façon de faire un don à BeeBase, veuillez s'il vous plaît visiter `https://sourceforge.net/projects/beebase/donate`.

## 1.3 Liste de discussion

Une liste de discussion spécifique à BeeBase se trouve sur `https://groups.google.com/g/beebase-bbs`. Venez nous rejoindre, vous êtes tous les bienvenus.

Veuillez envoyer les rapports de bogue ou les demandes d'évolution sur cette liste de discussion.

## 1.4 Mise en garde

## 1.5 Parties externes

BeeBase utilise des bibliothèques externes et d'autres matériels en fonction du système d'exploitation.

### 1.5.1 Versions Windows, Mac OS et Linux

BeeBase pour Windows, Mac OS et Linux utlisent Qt 5.12 ou plus, (copyright © The Qt Company). Qt est un framework multi-plateforme disponible sous Licences GPL et LGPLv3. Pour plus d'informations, consulter `https://www.qt.io`.

Sous Linux BeeBase utilise la version 3.24 (ou supérieure) du Gimp Toolkit (GTK+), copyright © The GTK+ Team. GTK est une boîte à outils permettant de développer des Interfaces Homme Machine (IHM) distribuée sous licence GNU Library General Public License (LGPL). Pour plus d'informations, consulter `https://www.gtk.org`.

Le script d'installation sur Windows pour BeeBase est basé sur le script d'installation de Gimp qui est sous copyright de Jernej Simoncic.

### 1.5.2 Version Amiga

```
                La version Amiga de BeeBase utilise


                    MUI - MagicUserInterface

            Copyright © 1992-2013, Stefan Stuntz


MUI est un système pour générer et maintenir des interfaces graphiques
utilisateurs. À l'aide d'un programme de préférences, l'utilisateur d'une
application a la capacité de l'adapter selon son goût personnel.


MUI est distribué sous forme de shareware. Pour obtenir la distribution complète
contenant un bon nombre d'exemples et plus d'informations sur l'enregistrement,
recherchez s'il vous plaît le fichier appelé « muiXXusr.lha » (XX représente le
dernier numéro de version) dans vos tableaux de liens locaux ou sur les
disques du domaine public

Si vous voulez vous enregistrer directement, sentez-vous libre d'envoyer


                    DM 30.-  ou  US$ 20.-

                            à

                        Stefan Stuntz
                    Eduard-Spranger-Straße 7
                        80935 München
                           GERMANY
```

`Support et enregistrement en ligne sont disponibles sur`

`http://www.sasg.com/`

Sur Amiga, BeeBase utilise NList.mcc, copyright © 2001-2015 NList Open Source Team. Voir `https://sourceforge.net/projects/nlist-classes` pour de plus amples informations ou la dernière version.

Sur Amiga, BeeBase utilise aussi BetterString.mcc, copyright © 2005-2015 BetterString Open Source Team. Voir `https://sourceforge.net/projects/bstring-mcc` pour de plus amples informations ou la dernière version.

Sur Amiga, BeeBase utilise également TextEditor.mcc, copyright © 2005-2015 TextEditor Open Source Team. Voir `https://sourceforge.net/projects/texteditor-mcc` pour de plus amples informations ou la dernière version.

Sur Amiga, BeeBase utilise aussi codesets.library, copyright © 2005-2015 codesets.library Open Source Team. Voir `https://sourceforge.net/projects/codesetslib` pour de plus amples informations ou la dernière version.

Quelques icônes utilisées dans BeeBase pour la distribution Amiga sont extraites du jeu d'icônes DefaultIcons et sont sous copyright © de Michael-Wolfgang Hohmann et Angela Schmidt.

# 2 Bienvenu sur BeeBase

BeeBase est une base de données relationnelle programmable avec une interface graphique pour Windows, Mac, Linux et Amiga.

BeeBase est adapté à la gestion de données structurées avec une sémantique claire pouvant être organisée en tables et champs. Il peut être considéré comme un environnement de développement d'applications dans lequel un concepteur d'application crée la structure de la base de données et l'interface utilisateur ; alors que les utilisateurs de l'application saisissent et traitent régulièrement les données.

Il existe de nombreuses applications pour BeeBase, notamment la gestion dadresses, de musique, de films, de collections de photos, de votre arbre généalogique ou de vos revenus et dépenses. Certains utilisateurs gèrent l'ensemble de leur entreprise à l'aide de BeeBase depuis la planification de projet jusquaux finances en passant par la création d'offres, la gestion d'articles et de pièces, la facturation.

La force de BeeBase réside dans son interface utilisateur graphique et ses capacités de programmation. Il vous permet de traiter les données de différentes manières, par ex. calculer automatiquement lors de la saisie de l'utilisateur, générer des rapports, importer et exporter de données, etc. Ce que vous pourriez utiliser par exemple pour calculer le montant total de vos revenus, ou le montant total du temps enregistré sur un CD,ou encore créer et imprimer automatiquement des publipostages pour vos clients.

BeeBase offre les caractéristiques suivantes :

- Accès partagé à de multiples instances d'un projet BeeBase sur le même ou différents ordinateurs.
- Nombre illimité de projets, tables, champs et enregistrements.
- Les champs peuvent être de type chaîne de caractères, mémo (texte multiligne), entier, réel, date, heure, booléen, liste de choix (sélection d'un élément dans une liste), référence (faire référence facilement à un enregistrement d'une autre table), bouton (pour exécuter des programmes BeeBase) et virtuel (valeur calculée à la volée).
- Le type chaîne de caractères peut également contrôler des listes de chaînes, fichiers et polices de caractères. Une chaîne de caractères peut se rapporter à une image externe qui sera affichée dans l'interface graphique.
- Chargement dynamique des enregistrements. Les enregistrements qui n'ont pas été consultés récents peuvent être déchargés de la mémoire (lorsque par exemple la mémoire vient à manquer).
- Programmation. Avec le langage de programmation facile et puissant de BeeBase, des tâches complexes peuvent être mises en application. Le langage inclut également un langage de requête de type SQL (SELECT FROM WHERE) pour une recherche confortable et rapide des données.
- Ordonnancement des enregistrements par n'importe quelle combinaison de champs.
- Fonctions de recherche et de filtrage flexibles et puissantes.
- Éditeur de requêtes permettant de taper et manipuler des requêtes SELECT FROM WHERE. Les requêtes peuvent être sauvées et les résultats imprimés.
- Journalisation facultative des modifications (ajout, suppression et modification des enregistrements) dans une table journal système.

- Importation et exportation aisée.
- Documentation complète comprenant un manuel utilisateur et une référence de programmation en HTML et PDF (plus AmigaGuide sur Amiga).
- La version Amiga fournit une interface ARexx puissante pour accéder à BeeBase à partir d'autres programmes. L'interface ARexx offre un mécanisme de transaction semblable à d'autres bases de données relationnelles.
- Indépendance du système d'exploitation. BeeBase est disponible pour Windows, Mac OS, Linux et Amiga. Le code source est disponible en tant que projet SourceForge.

BeeBase est le successeur de MUIBase, dont la première version a vu le jour en 1994 sur Amiga.

# 3 Pour commencer

Ce chapitre décrit le matériel et les logiciels nécessaires pour exécuter BeeBase sur votre ordinateur, l'installer, le mettre à jour, démarrer et quitter.

## 3.1 Installer BeeBase sur Windows

BeeBase pour Windows fonctionne sur les ordinateurs compatibles Intel qui utilisent les systèmes Windows 10 [et ultérieurs] (64 bit).

L'installateur de BeeBase est distribué sous la forme d'un exécutable Windows, par exemple : '`BeeBase-1.0-setup-x64.exe`' que vous pouvez télécharger sur la page d'accueil de BeeBase `https://beebase.sourceforge.io`. Pendant l'installation, de nombreuses questions vous seront demandées à propos de l'emplacement d'installation de BeeBase, et de sa configuration. Au cas où vous auriez une version précédente de BeeBase installée sur votre ordinateur, la procédure d'installation permet la mise à jour vers la nouvelle version.

Après une installation réussie, le menu '`Démarrer`' de Windows devrait contenir un nouveau raccourci vers BeeBase. Vous pouvez maintenant supprimer le fichier d'installation de BeeBase, il n'est plus nécessaire.

## 3.2 Installer BeeBase sur Mac OS

BeeBase pour Mac OS prend en charge macOS Big Sur (version 11) et les versions suivantes du système d'exploitation Mac OS. Le logiciel est distribué au format binaire universel qui contient des binaires 64 bits pour les processeurs Intel et Arm.

Une image disque de BeeBase, c.-à-d. '`BeeBase-1.0.dmg`', peut être téléchargée depuis le site `https://beebase.sourceforge.io`.

Après le téléchargement et l'ouverture de l'image, glissez-déposez '`BeeBase.app`' dans le dossier '`Applications`' de l'ordinateur.

## 3.3 Installer BeeBase sur Linux

Vous pouvez installer et lancer BeeBase sur un ordinateur compatible Intel fonctionnant avec le système d'exploitation Linux.

BeeBase pour Linux est distribué à la fois sous la forme d'un paquet Debian (p. ex. '`beebase_1.0_arm64.deb`' pour Ubuntu et Debian) et sous la forme d'une archive RPM (p. ex. '`BeeBase-1.0-fc34.aarch64.rpm`' pour Fedora).

Les deux versions peuvent être téléchargées à partir de la page d'accueil de BeeBase `https://beebase.sourceforge.io`.

Habituellement BeeBase s'installe ou se met à jour lui-même automatiquement après le téléchargement sur votre ordinateur. Dans le cas où l'installation ne se lancerait pas automatiquement, les commandes suivantes exécutées les droits du super utilisateur (root) depuis un interpréteur de commande installent ou mettent à jour BeeBase sur un système Linux sous Debian :

```
dpkg --install beebase.deb
apt-mark hold beebase.deb
```

où *beebase.deb* est le paquet Debian téléchargé.

Sur un système dérivé de Redhat (RPM) les commandes sont :

```
rpm -Uvh BeeBase.rpm
```

où *BeeBase.rpm* est le fichier RPM de BeeBase téléchargé,

Après une installation réussie vous devriez voir un nouvel élément dans le menu '`Applications - Office`' de votre bureau Linux.

## 3.4 Installer BeeBase sur Amiga

BeeBase pour Amiga fonctionne sur la version 3.0 (ou supérieure) de l'OS Amiga et exige un processeur 68020 ou supérieur. En outre, la version 3.8 (ou supérieure) de MUI doit être présente sur votre système. Un minimum de 4 Mo de RAM et de 10 Mo d'espace libre sur votre disque dur sont nécessaires. On m'a signalé que le binaire m68k de BeeBase pour Amiga se lance également sur UAE, MorphOS et Amiga OS4.

Des exécutables natifs pour MorphOS, Amiga OS4 et i386/x86_64 AROS sont disponibles dans l'archive distribuée.

BeeBase est distribué sous forme d'une archive lha (p. ex. '`BeeBase-1.0.lha`') peuvent être téléchargées à partir de la page d'accueil de BeeBase `https://beebase.sourceforge.io`. Pour l'installer BeeBase sur votre ordinateur, vous devez décompacter ces archives dans un répertoire provisoire. Ne pas les décompacter dans le répertoire cible !

Double cliquez sur le script d'installation de BeeBase `Install-BeeBase` et suivez les instructions. Le script demande le répertoire où le logiciel doit être copié. N'indiquez pas le chemin d'accès du répertoire où vous avez décompacté l'archive de BeeBase. Le script est également capable de mettre à jour une installation existante de BeeBase, il suffit pour cela de lui fournir le chemin d'accès du répertoire où vous avez précédemment installé BeeBase.

Après une installation réussie, vous trouverez sur votre système le (nouveau) répertoire contenant le programme BeeBase avec tous les fichiers nécessaires et quelques échantillons de projets. De plus, une assignation `BeeBase:` se rapportant à ce répertoire a été rajoutée sur votre fichier système `S:User-Startup`. Vous pouvez enlever les archives lha et les fichiers temporaires, ils ne sont plus nécessaires.

## 3.5 Mise à jour d'une version précédente

En installant BeeBase sur Linux, Windows ou Amiga, vous pouvez mettre à jour la version précédente ou la réinstaller comme décrit ci-dessus. Pendant une nouvelle installation, tous les fichiers nécessaires sont remplacés. Ce qui inclus tous les échantillons de projets dans le répertoire `Démos` . Il n'est donc pas recommandé de placer vos propres projets dans ce répertoire, ni d'employer l'un de ces projets pour gérer vos propres données : elles risqueraient d'être écrasées pendant la réinstallation ou la mise à jour de BeeBase.

Il est recommandé de gérer vos propres projets en les plaçant dans un répertoire séparé du répertoire d'installation de BeeBase.

Si votre système contient une version de MUIbase, il est recommandé de la supprimer, afin que les associations de fichiers soient correctement affectées à BeeBase.

## 3.6 Démarrer BeeBase

BeeBase peut être démarré à partir de votre environnement graphique ou à partir de la ligne de commande. Sur Windows vous trouverez BeeBase dans le menu '`Démarrer`'. Sur Mac OS vous pouvez lancer BeeBase avec le '`Finder`' depuis le répertoire '`Applications`'. Si vous utilisez Gnome ou KDE sur Linux, vous pouvez démarrer BeeBase en choisissant l'entrée correspondante dans le menu '`Applications - Office`'. Sur le Workbench (Amiga), vous pouvez double-cliquer sur l'icône BeeBase ou sur l'icône du projet BeeBase qui se chargera automatiquement après avoir démarré BeeBase (vous pouvez également sélectionner plusieurs projets de BeeBase à l'aide de la touche shift puis double-cliquer sur le dernier).

En démarrant BeeBase à partir de la ligne de commande Windows, Linux ou Amiga vous pouvez inclure des arguments dans la ligne de commande et charger des projets optionnels. Il y a deux manières pour démarrer BeeBase à partir d'une commande shell :

```
BeeBase [projet1 ...]
BeeBase -n
```

La première forme démarre BeeBase et charge les projets optionnels spécifiés par leurs noms de fichiers *project1* .... La seconde forme démarre BeeBase sans interface graphique. Ce qui peut être utile pour démarrer BeeBase comme serveur en tâche de fond et pour accéder à son interface externe (par exemple ARexx sur Amiga, voir Chapitre 17 [Interface ARexx], page 167).

## 3.7 Quitter BeeBase

Pour quitter BeeBase sélectionnez l'entrée du menu '`Projet - Quitter`' ou fermez tous les projets ouverts.

## 3.8 Nomenclature de fichiers sur Windows, Mac OS et Linux

BeeBase pour Windows, Mac OS et Linux a quelques conventions spéciales pour les noms de fichiers qu'on ne trouve habituellement pas dans d'autres logiciels (sur ces systèmes).

Lorsque BeeBase démarre, une variable d'environnement '`BeeBase`' est créée. Elle indique le répertoire d'installation de BeeBase. Sur Windows c'est le répertoire spécifié durant l'installation du logiciel. Sur Mac OS, c'est le répertoire '`/Applications/BeeBase.app/Contents/Resources/share/BeeBase`'. Sur Linux, c'est le répertoire `/usr/share/BeeBase`.

Lors de l'interprétation des noms de fichiers (entrés par l'utilisateur ou ceux présents dans cette documentation), un nom de fichier est analysé à la recherche de variables d'environnement. Si un nom de fichier contient le signe dollar '`$`' alors les caractères suivants jusqu'au premier caractère non alphanumérique sont interprétés comme étant une variable d'environnement et, si cette variable d'environnement est positionnée, cette partie du nom de fichier est remplacée par le contenu de la variable d'environnement (autrement, la partie du nom de fichier, y compris le signe '`$`', reste inchangée). Vous pouvez mettre des parenthèses autour de la variable d'environnement si elle contient des caractères non alphanumériques. Par exemple `$HOME/fichier` est interprété comme le chemin d'accès où `$HOME` a été remplacé par le nom de votre répertoire personnel.

En outre, les ‘`assignations`’ style Amiga sont manipulées de la façon suivante. Si un nom de fichier contient deux points ‘`:`’ alors, tout ce qui se trouve avant les deux points est traité comme une variable d'environnement (appelée ‘`assignation`’) et est remplacé par son contenu si la variable d'environnement est positionnée. Les ‘`Assignations`’ devraient être au moins composées de 2 caractères, pour ne pas les confondre avec les noms des volumes Windows.

Ces conventions permettent, par exemple, de se référer à la base de données cinématographique d'exemple qui se trouve à l'intérieur du répertoire d'installation de BeeBase en utilisant `BeeBase:demos/Movie.bbs`.

Un autre exemple : lorsque vous avez positionné une variable d'environnement ‘`EXTERNAL_DATA`’ pointant vers un répertoire où vous stockez les données externes d'un de vos projets (comme des images, etc. ). Vous pouvez alors accéder aux données à l'intérieur de ce répertoire en utilisant `EXTERNAL_DATA:un-fichier` et stocker de tels noms de fichier dans le projet de base de données.

# 4 Tutoriel

Création d'une base généalogique.

Ce chapitre contient une brève formation décrivant l'utilisation basique de BeeBase. Au fil de ces travaux dirigés un petit projet permettant la gestion de son arbre généalogique est développé. Après avoir appliqué toutes les étapes de ce cours, vous pourrez trouver le projet qui en résulte '`FamilyTree.bbs`' dans le répertoire `Demos` de votre installation de BeeBase.

## 4.1 Comment fonctionne BeeBase

On peut dire que BeeBase fonctionne dans deux modes différents : l'édition d'enregistrements et l'édition de structure.

Le mode d'édition d'enregistrements vous permet d'ajouter, de changer ou de supprimer des enregistrements.

Le mode d'édition de structure vous permet de modifier la mise en page de votre base de données ainsi que les tables et champs qu'elle contient.

En outre, il y a l'éditeur de programme où vous saisissez les fonctions du programme qui seront exécutées automatiquement lors de l'entrée des données ou explicitement lors d'un clic sur un bouton du programme.

## 4.2 Pour commencer un Projet : l'éditeur de structure

Pour créer une base de données, vous devez d'abord définir son contenu. Pour cela, il faut ouvrir l'éditeur de structure en choisissant '`Éditeur de structure`' dans le menu '`Projet`'. Voici les trois différentes sections que vous y trouverez :

'`Tables`' Ajouter, changer ou supprimer les tables de votre projet.

'`Champs`' Ajouter, changer ou supprimer les champs de la table sélectionnée.

'`Affichage`'
Indiquer la disposition graphique de votre base de données, c.-à-d. la façon dont elle devra être affichée à l'utilisateur.

## 4.3 Ajouter une table

Tout d'abord nous avons besoin d'une table, appuyez sur le bouton '`Nouveau`' situé sous la liste de la section '`Table`'. Vous verrez alors une fenêtre s'ouvrir pour vous demander de saisir quelques données :

'`Nom`' Le nom de la table. Le nom doit commencer par une lettre majuscule et peut se composer d'un maximum de 20 caractères. Le nom peut être changé plus tard. Dans ce cours nous définissons le nom '`Personne`' puisqu'il va contenir les noms de toutes les personnes dans cette base de données.

'`Nombre d'enregistrements`'
Une table peut soit contenir un seul enregistrement, soit un nombre illimité. Dans notre cas il faut le positionner « illimité » puisque nous allons ajouter plus d'une personne.

‘Déclencheurs’
: L'ajout et la suppression des enregistrements peuvent être contrôlés par des fonctions utilisateur. C'est ici que vous pouvez indiquer la fonction à exécuter. Puisque nous n'avons encore écrit aucune fonction utilisateur, nous laissons les champs vides.

Quand c'est fait, cliquez juste sur le bouton ‘OK’ et vous aurez votre première table, ‘Personne’.

## 4.4 Ajouter un champ

Maintenant nous avons besoin d'un champ de type chaîne pour cette table, appuyez sur le bouton ‘Nouveau’ dans la section Champs. Les champs ont également besoin de quelques réglages :

‘Nom’
: Comme pour une table : la première lettre doit être en majuscule et peut se composer au maximum de 20 caractères. Nous définissons le nom de ce champ à ‘Nom’ car il contiendra les noms des personnes que nous sommes sur le point d'ajouter.

‘Type’
: Ici nous choisissons le type de champ désiré. Il en existe un certain nombre, mais pour ce champ nous choisirons d'utiliser le type Texte.

‘Longueur maximum’
: Vous pouvez définir le nombre maximum de caractères qu'un utilisateur pourra écrire dans la chaîne. Choisissons 30.

‘Valeur initiale’
: Il est possible qu'un champ utilise une valeur initiale à chaque nouvel enregistrement que vous ajoutez, entrez ici cette valeur. Laissez cette ligne vide pour notre exemple.

‘Déclencheur’
: Un champ peut également déclencher l'exécution d'une fonction utilisateur. Par exemple, lorsque vous entrez un nom, vous pouvez avoir une fonction utilisateur qui vérifie si ce nom existe déjà. Nous laissons ce champ vide.

## 4.5 Afficher le projet

Après avoir cliqué sur le bouton ‘OK’ vous pourrez noter quelques changements dans la section d'affichage. Choisissez ‘Fenêtre Principale’ dans la liste déroulante en haut de la section d'affichage. Vous verrez ce que la fenêtre principale propose : actuellement seulement la table ‘Personne’. Si vous revenez de nouveau sur ‘Fiche de table’, vous pourrez voir comment la table ‘Personne’ se présente. Actuellement elle affiche un onglet avec un champ.

Double-cliquez maintenant sur l'‘Personne’ en haut de la liste, dans la section d'affichage ; une fenêtre doit s'ouvrir vous permettant de définir la façon dont cet onglet doit être affiché :

‘Titre’
: Le titre d'une table peut être différent de son nom. Notre table s'appelle ‘Personne’ mais ici nous pourrions aussi bien mettre ‘C'EST LA TABLE PERSONNE !’ si nous préférions.

‘`Arrière-plan`’
: L’arrière-plan peut être modifié selon votre goût.

‘`Gadgets`’ Ici, nous pouvons définir les gadgets que nous désirons voir sur l’onglet.

Appuyez sur le bouton ‘`OK`’ et double-cliquez sur ‘`Nom`’ dans la liste de la section d’affichage. La fenêtre ‘`affichage du champ`’ apparaît, elle contient les réglages sur la façon dont le champ ‘`Nom`’ va être affiché.

‘`Titre`’ Les mêmes que pour l’onglet. La chaîne que vous entrez ici est ce que vous voyez réellement lorsqu’on est en mode édition d’enregistrements.

‘`Raccourci clavier`’
: Permet de placer une lettre qui peut être utilisée pour sauter vers ce champ, lorsqu’on est en mode édition d’enregistrements. Pour sauter vers un champ vous devez maintenir la touche ***`Alt`*** (Windows, Mac OS et Linux) ou la touche ***`Amiga`*** et appuyer sur la lettre.

‘`Début`’ Cochez la case début pour sauter vers ce champ toutes les fois qu’un nouvel enregistrement est ajouté. Dans notre cas nous voulons toujours, ou la majeure partie du temps, présenter d’abord le premier nom dans un nouvel enregistrement. Donc, cochez la case début.

‘`Lecture seule`’
: À cocher si le champ est en lecture seule. Laissez tel quel.

‘`Poids`’ Décide dans quelle proportion de l’espace disponible ce champ devra être visible (en concurrence avec d’autres champs). Par exemple, si trois chaînes de 50 caractères se trouvent dans une fenêtre n’offrant qu’une place limitée de 100 caractères, ce nombre décidera combien d’espaces la chaîne obtiendra par rapport aux autres. Le laisser à 100.

‘`Arrière-plan`’
: Les mêmes que pour l’onglet.

‘`Info bulle`’
: Ici vous pouvez écrire n’importe quel texte qui vous paraît utile pour un utilisateur. Une infobulle apparaît lorsque la souris est maintenue quelques secondes au-dessus d’un champ. Saisissez ceci ‘`Si vous avez besoin d’aide, appelez l’auteur au 112`’.

Quittez l’éditeur de structure (choisissez ‘`Quitter l’éditeur de structure`’ dans le menu ‘`Projet`’) et vous serez de nouveau en mode édition d’enregistrements où vous verrez à quoi ressemble vraiment la base de données. Vous verrez le titre qui correspond à la chaîne que vous avez choisie dans la section d’affichage de l’onglet. Le compteur d’enregistrement devrait indiquer ‘`#0/0`’ puisque nous n’avons pas encore ajouté d’enregistrement. À la suite, se trouve un bouton filtre et deux boutons flèches. Au-dessous, vous devriez avoir le champ ‘`Nom`’ avec le texte que vous avez pu écrire dans la section d’affichage de ce champ. Si dans la section d’affichage, vous n’avez changé aucun texte, alors l’onglet devrait s’appeler ‘`Personne`’, et le champ de type chaîne ‘`Nom`’. Déplacez la souris au-dessus de la chaîne champ ‘`Nom`’ et laissez-la pendant quelques secondes. Si vous avez écrit quelque chose dans la bulle d’aide, ce texte devrait paraître dans une fenêtre flottante.

## 4.6 Ajouter deux champs de type référence

Maintenant nous ajouterons deux champs référence. Les champs référence sont un peu différents des autres champs. Car leur nom pourrait impliquer qu'ils se réfèrent à des enregistrements. Ce sera plus compréhensible, lorsque dans un moment vous l'essayerez par vous-même.

Entrez dans l'éditeur de structure et ajoutez deux autres champs à la table '`Personne`'. Cliquez sur le bouton '`Nouveau`' dans la section champ, la fenêtre nouveau champ s'ouvre, nommez le '`Pere`', et changez le type en '`Référence`'. Un champ Référence ne dispose que d'un réglage :

'`Faire référence à`'
: Indique à quelle table le champ se rapporte. Il devrait déjà pointer vers '`Personne`'. Laissez-le comme ça et appuyez sur le bouton '`Ok`'.

Ajoutez un autre champ en cliquant sur le bouton '`Nouveau`' dans la section Champs, la fenêtre '`Nouveau champ`' s'ouvre. Nous nommons ce champ '`Mere`'. Positionnons son type de champ à une référence vers la table '`Personne`'.

Comme vous avez pu le noter, il y a maintenant trois champs dans la section Affichage. Cliquez sur '`Pere`' puis sur les boutons haut et bas placés juste à gauche. Ce qui aura pour effet de changer l'affichage et le positionnement de '`Pere`' dans le mode édition d'enregistrements. Mettez '`Pere`' en haut, '`Nom`' au milieu, et '`Mere`' en bas.

Après, nous indiquerons quel contenu les champs référence '`Pere`' et '`Mere`' devraient afficher dans les enregistrements référencés. Double-cliquez sur '`Pere`' dans la section d'affichage puis cliquez sur '`Extras`'. Ici nous choisissons d'afficher le champ '`Nom`', appuyez sur le bouton '`Ok`' et répétons le procédé pour le champ '`Mere`'.

## 4.7 Ajouter des enregistrements

Maintenant nous devons ajouter quelques enregistrements. Quittez l'éditeur de structure et entrez dans le mode édition d'enregistrements. Pour ajouter un nouvel enregistrement, choisissez '`Nouvel enregistrement`' dans le menu '`Table`' (sur Linux et Windows vous trouverez ce menu en maintenant pressé le bouton droit de la souris à l'intérieur de la fiche de table). Le curseur devrait maintenant sauter automatiquement au champ que nous avons placé un peu plus tôt comme étant au '`Début`' dans la section d'affichage de l'éditeur de structure. Ajoutez deux enregistrements, un avec le nom de votre père dans '`Nom`' et un autre avec le nom de votre mère dans '`Nom`'. Ensuite, vous ajoutez un autre enregistrement avec votre propre nom dans '`Nom`'.

Maintenant il est temps de comprendre les champs Référence. Cliquez sur le bouton de sélection dans '`Pere`' et vous obtiendrez la liste de tous les enregistrements auquel ce champ Référence pourrait se rapporter. Choisissez le nom de votre père puis, suivre le même cheminement pour le bouton de sélection dans mère.

Maintenant, vous devriez avoir trois enregistrements, vous, votre père et votre mère. Dans votre enregistrement, le nom de votre père devrait être évidemment visible en haut dans '`Pere`' et le nom de votre mère devrait être en bas dans '`Mere`'. Vous pouvez passer en revue les enregistrements en appuyant sur *`Alt`* ainsi que sur les touches *`Haut`* et *`Bas`*.

Vous vous dîtes : « Mais hé, mes parents aussi ont/avaient des parents ! » Aussi, ajoutons encore quatre enregistrements, la troisième génération. Ajoutez juste les enregistrements un

par un en écrivant leurs noms dans ‘Nom’. Si vous ne vous rappelez pas leurs noms écrivez juste ‘le père de mon père’, ‘le père de ma mère’ ou toute autre formule équivalente du même style. Vous pourrez alors naviguer dans tous les enregistrements et placer dans ‘Pere’ et ‘Mere’ les valeurs adéquates. Une fois cela fait, vous devriez avoir au moins sept enregistrements, votre enregistrement, l'enregistrement de vos parents et l'enregistrement de vos grands-parents.

## 4.8 Filtrer des enregistrements

Puisque nous avons maintenant quelques enregistrements avec lesquels nous pouvons travailler, nous pourrions essayer la fonction de filtrage. Le filtre peut trier les enregistrements que vous ne voulez pas montrer, ils resteront dans la base de données mais ils ne seront simplement pas affichés.

Pour éditer le filtre choisissez ‘Changer le filtre’ dans le menu ‘Table’. Une fenêtre comportant des opérateurs apparaîtra, vous permettant de définir les conditions qu'un enregistrement doit remplir pour être affiché.

Dans ce petit exemple, nous utiliserons la commande LIKE, qui vous laisse comparer un champ avec un motif. Appuyez une fois sur le bouton LIKE vers la droite et double-cliquez sur ‘Nom’ dans la liste à gauche et le (LIKE Nom ) devrait apparaître dans la chaîne juste au-dessus des boutons ‘Ok’ et ‘Annuler’. Ensuite vous écrivez "*a*" de manière à ce que le texte entier devienne (LIKE Nom "*a*"). Ceci signifie que BeeBase devrait afficher tous les enregistrements qui contiennent la lettre ‘a’ n'importe où dans le ‘Nom’.

Appuyez sur le bouton ‘Ok’ et vous pourrez noter que les enregistrements ne contenant pas de ‘a’ dans leur ‘Nom’, ne seront plus visibles. Puisque la lettre ‘a’ est très courante dans la plupart des langues et noms, tous les enregistrements pourraient encore être affichés, mais vous pouvez essayer d'autres lettres pour faire fonctionner le filtre de façon plus visible.

Comme expliqué précédemment, le bouton ‘F’ sur l'onglet indique si le filtre est actif ou pas. Enfin, lorsque vous aurez fini vos essais, quittez le filtre de sorte que tous les enregistrements soient visibles.

## 4.9 Interroger des enregistrements

Maintenant que nous avons un peu joué avec la fonction de filtrage, nous nous tournons vers le dispositif de requête de BeeBase. Des requêtes peuvent être utilisées pour afficher les données d'une base selon certains critères.

Choisissez ‘Requêtes’ dans le menu ‘Programme’ ; l'éditeur de requête s'ouvre. Maintenant une fenêtre avec quelques boutons en haut et deux plus grandes zones au-dessous apparaissent. La chaîne en haut à gauche permet d'écrire le nom de la requête que vous voulez créer.

‘Exécuter’
: Compile et lance la requête.

‘Imprimer’
: Imprime le résultat de la requête dans un fichier ou sur votre imprimante.

‘Charger et Enregistrer’
: Vous permet de charger et enregistrer chacune des requêtes.

La première grande zone permet d'écrire la requête. La deuxième grande zone permet d'afficher le résultat.

Maintenant produisons une liste de toutes les personnes que nous avons filtrées précédemment. Écrire : '`Personnes avec un A dans leur nom`' dans la chaîne en haut à gauche. C'est le titre pour cette requête. Dans la grande zone supérieure, taper :

```
SELECT Nom FROM Personne WHERE (LIKE Nom "*a*")
```

Maintenant si vous lancez cette requête en appuyant sur le bouton '`Exécuter`' il produira une liste de toutes les personnes avec la lettre '`A`' dans leur nom. Essayez de changer la lettre pour voir différents résultats.

Il est temps de présenter la commande `AND`. Appuyez sur le bouton de sélection juste à gauche du bouton '`Exécuter`' dans l'éditeur de requêtes. Choisissez alors '`Nouveau`' et nommez-la '`Personnes dont le nom contient les lettres A et S`'. Écrivez alors

```
SELECT Nom FROM Personne WHERE
(AND (LIKE Nom "*a*") (LIKE Nom "*s*"))
```

Notez que nous employons toujours la commande `LIKE` pour le choix des enregistrements contenants les lettres '`A`' ou '`S`' dans leurs noms, mais la commande `AND` exige que LES DEUX critères `LIKE` soient vérifiés. Par conséquent, seuls les enregistrements comportant les DEUX lettres '`A`' et '`S`' dans leur nom sont affichés lorsque la requête est exécutée.

## 4.10 Ajouter une table avec un mémo et un champ de type bouton

Il y a deux manières de choisir et de montrer la base de données. Une autre manière d'afficher les données consiste à écrire un programme. Pour afficher les données, nous pouvons utiliser un type de champ appelé Mémo.

Entrez dans l'éditeur de structure et appuyez sur le bouton '`Nouveau`' dans la section table. Nommez la nouvelle table '`Controle`' et placez son nombre d'enregistrements à '`Exactement un`'. Pressez et maintenez le bouton de la souris sur la nouvelle table, puis déplacez le curseur juste un peu au-dessus du milieu de la table '`Personne`' et enfin relâchez le bouton de la souris. Dans la section table, '`Controle`' devrait maintenant être en haut, suivi de '`Personne`'.

Assurez-vous que la table '`Controle`' est sélectionnée, et appuyez sur le bouton '`Nouveau`' dans la section Champs. Placez le type du nouveau champ sur '`Mémo`' et donnez-lui le nom '`Resultat`'. Cliquez sur le bouton '`Ok`' puis ajoutez un autre champ à la table '`Controle`' en appuyant de nouveau sur le bouton '`Nouveau`' dans la section Champs. Cette fois, choisissez le type '`Bouton`' et appelez le '`Ascendance`'.

Pour améliorer l'aspect de la base de données, dans la section d'affichage, cliquez une fois sur '`Ascendance`' ainsi que sur le bouton '`haut`'.

## 4.11 Programmation d'une ascendance avec BeeBase

Maintenant nous avons un bouton qui peut exécuter un programme et un mémo pour y afficher des données. Il est donc temps d'entrer dans l'éditeur de programme. Pour cela, il faut choisir '`Editer`' à partir du menu '`Programme`'. L'éditeur de programme à trois boutons :

‘`Compiler & Fermer`’
: Il fait juste cela. Il compile le programme et sort de l’éditeur de programme.

‘`Compiler`’
: Compile les programmes mais reste dans l’éditeur de programme.

‘`Rétablir`’
: Rétablit tous les changements pour revenir à l’état initial lors de votre entrée dans l’éditeur de programme.

Toutes les fonctions de programme que vous écrirez résideront dans cette fenêtre, nous devrons les distinguer. Dans BeeBase c’est la commande `DEFUN` qui le fait. Tout ce qui est entre les deux parenthèses fera partie, dans cet exemple, de la fonction `ascendance` :

```
(DEFUN ascendance ()

; C'est la parenthèse de la fin de DEFUN
)
```

À cet effet nous écrivons maintenant la première fonction qui produira, dans la base de données, un arbre généalogique de la personne courante et plaçons le résultat dans le champ ‘`Resultat`’. Cette fonction `ascendance` comporte en fait trois fonctions :

- `ascendance` qui remplit ‘`Controle.Resultat`’ avec le résultat de l’appel à une autre fonction.
- `getascendance` qui rassemble l’ascendance dans une liste.
- `ascendance2memo` qui convertit cette liste en mémo.

```
; Le programme ascendance

(DEFUN ascendance ()
    (SETQ Controle.Resultat (ascendance2memo (getascendance Personne NIL) 0 3))
)



; Le programme getascendance

(DEFUN getascendance (personne:Personne niveau:INT)
    (IF (AND personne (OR (NULL niveau) (> niveau 0)))
        (LIST personne.Nom
            (getascendance personne.Pere (1- niveau))
            (getascendance personne.Mere (1- niveau))
        )
    )
)



; Le programme ascendance2memo

(DEFUN ascendance2memo (ascendance:LIST indent:INT niveau:INT)
    (IF (> niveau 0)
```

```
            (+
                (ascendance2memo (NTH 1 ascendance) (+ indent 8) (1- niveau))
                (IF pedigree (SPRINTF "%*s%s\n" indent "" (FIRST ascendance)) "\n")
                (ascendance2memo (NTH 2 ascendance) (+ indent 8) (1- niveau))
            )
            ""
        )
    )
```

En écrivant ce programme, assurez-vous que toutes les parenthèses se trouvent où elles devraient être. Un nombre de paranthèses inexact est l'erreur la plus commune lorsque BeeBase compile votre programme. Le message d'erreur de BeeBase serait alors probablement '`Erreur syntaxique`' . Cliquez sur le bouton '`Compiler & fermer`' et la fenêtre se ferme, ce qui signifie qu'il n'y a eu aucune erreur lors de la compilation.

Ne vous inquiétez pas trop si au début, vous ne comprenez pas toutes les fonctions. Il y a un chapitre complet (voir Chapitre 16 [Programming BeeBase], page 81) contenant la liste de toutes les fonctions avec leur description détaillée.

Maintenant nous avons un programme à exécuter, mais d'abord nous devons assigner la fonction programmée au bouton '`Ascendance`'. Pour cela, il faut entrer dans l'éditeur de structure, choisir '`Controle`' dans la section Table et double-cliquer sur le champ '`Ascendance`' dans la section Champs. Cliquez alors le bouton de sélection '`Déclencheur`'. Dans cette liste, toutes les fonctions de votre programme seront énumérées, à ce moment, il devrait y avoir trois fonctions : `ascendance`, `getascendance` et `ascendance2memo`. Double-cliquez sur `ascendance` pour que le bouton '`Ascendance`' déclenche cette fonction, puis appuyez sur le bouton '`Ok`' et quittez l'éditeur de structure.

Maintenant si tout est fait correctement, un clic sur le bouton '`Ascendance`' affichera l'ascendance de la personne sélectionnée. Essayez de changer la personne pour voir différentes ascendances.

## 4.12 Programmation de la liste des enfants d'une personne avec BeeBase

Le prochain ajout sur cette base de données exige encore plus d'enregistrements, vous devriez ajouter vos frères et soeurs. Si vous n'en avez pas, écrivez '`Ma soeur fictive 1`', '`Mon frère fictif 1`' qui naturellement devront être placés de façon à avoir les mêmes parents que vous.

Après cela, allez dans l'éditeur de programme et écrivez le nouveau programme :

```
; Le programme enfants compte le nombre d'enfants d'une personne.
; D'abord nous définissons les variables, p. ex. « noms » pour les noms des enfants.

(DEFUN enfants ()
    (LET ( (noms "") (count 0) (parent Personne) )

    ; Pour tous les enregistrements de la table Personne, faire :
    ; Si la variable parent apparaît comme père ou mère
    ; dans un ou plusieurs enregistrements alors :
    ;   ajoutez le nom à la variable noms
    ;   incrémentez le compte de 1.
```

```
        (FOR ALL Personne DO
            (IF (OR (= parent Pere) (= parent Mere))
                (
                    (SETQ noms (+ noms Nom "\n"))
                    (SETQ nombre (1+ nombre))
                )
            )
        )

        ; Ensuite nous écrivons le résultat dans Controle.Resultat.
        ; Si la personne sélectionnée n'a aucun enfant, écrire une chaîne.
        ; Si elle a des enfants alors écrire une autre chaîne.

        (SETQ Controle.Resultat
            (+ Personne.Nom (IF (> nombre 0)
                (+ " est fier d'être le parent de " (STR nombre) " enfant(s).")
                " n'a aucun enfant (pour le moment :-)."
            ))
        )


        ; Si les parents ont des enfants alors ajouter leurs noms.

        (IF (<> nombre 0)
            (SETQ Controle.Resultat
                (+ Controle.Resultat "\n\n"
                    (IF (= nombre 1)
                        "Le nom de l'enfant est :"
                        "Les noms des enfants sont :"
                    )
            "\n\n"
                    noms
                )
            )
        )

        ; Parenthèse de fin de la commande LET.
        )

    ; Parenthèse de fin de DEFUN enfants
    )
```

Pour créer des variables, nous utilisons la commande `LET`. Les variables créées avec la commande `LET` sont locales et seulement visibles entre les parenthèses ouvrante et fermante de cette commande '`LET`'. Ainsi, toutes les commandes voulant accéder à ces variables doivent être placées à l'intérieur des parenthèses.

Tout ce dont nous avons besoin pour exécuter notre programme est un nouveau bouton programme. Par conséquent, entrez dans l'éditeur de structure et ajoutez un champ de type bouton dans la table '`Controle`'. Appelez-le '`Enfants`' et choisissez `enfants` comme fonction à déclencher.

Pour apporter un certain ordre dans la fiche de la table '`Controle`' il est maintenant temps de présenter les groupes. Tous les objets peuvent être ordonnés dans des groupes alignés verticalement ou horizontalement.

Dans la section d'affichage, cliquez sur '`Ascendant`' et Maj-cliquez sur '`Enfants`'. Ensuite cliquez sur le bouton '`Groupe`' sur la gauche. Maintenant les deux boutons de programme seront rassemblés et alignés verticalement dans un groupe. Cependant, comme nous voulons qu'ils soient alignés horizontalement, double-cliquez sur '`VGroup`' qui est apparu dans la section d'affichage. Une fenêtre s'ouvrira qui vous permettra modifier les réglages de ce groupe. Saisissez '`Programmes`' comme titre et cochez le bouton '`Horizontal`'.

À ce moment nous pouvons cacher le nom du champ '`Resultat`' dans la table '`Controle`'. Double-cliquez sur '`Resultat`' dans la section d'affichage et videz le champ titre. Ceci affichera le champ '`Resultat`' sans aucun titre.

Pour rendre les choses plus faciles, si nous ajoutons plus de programmes ou de champs dans la table '`Controle`', nous devrions placer '`Resultat`' et le groupe '`Programmes`' dans un groupe vertical. Soyez certain que vous avez seulement sélectionné '`Programmes`' et '`Resultat`' avec de cliquer sur '`Groupe`'. Cela place '`Programmes`' et '`Resultat`' dans un groupe vertical.

Quittez l'éditeur de structure et jetez un coup d'oeil au résultat. Appuyez sur le bouton '`Enfants`' pour voir le nombre et le nom des enfants de la personne courante.

Cet exemple peut très bien être étendu dans un programme généalogique complet. Les seules vraies limites sont votre imagination et la taille de votre disque dur.

# 5 Concepts de base

Avant de créer votre propre base de données et de commencer la saisie de données, vous devez vous familiariser avec certaines notions de base de BeeBase.

## 5.1 Projets

Un projet BeeBase se compose de tous les éléments dont vous avez besoin pour gérer vos données. Cela va de l'interface graphique aux programmes écrits pour le projet en passant par la saisie des données.

Un projet peut être chargé, sauvegardé ou effacé du disque. Toute modification d'un projet n'est effectuée qu'en mémoire. Vous pouvez revenir à tout moment à l'état de la dernière sauvegarde en rechargeant le projet.

BeeBase est capable de gérer de multiples projets en parallèle. Ainsi il n'est pas nécessaire de démarrer BeeBase une deuxième fois pour charger un autre projet.

De plus, plusieurs instances de BeeBase peuvent accéder à un même projet. Plusieurs instances peuvent lire un projet mais une seule peut le modifier. Ceci devrait fonctionner au travers d'un réseau lorsque le projet est disponible sur un lecteur partagé.

## 5.2 Tables

BeeBase organise les données en tables. Une table est constituée de lignes et de colonnes : une ligne correspond à un enregistrement alors qu'une colonne correspond à un champ.

Voici un exemple de structure pour un carnet d'adresses :

```
Nom               | Rue                 | Ville
------------------|---------------------|------------------------
Steffen Gutmann   | Wiesentalstr. 30    | 73312 Geislingen/Eybach
Charles Saltzman  | University of Iowa  | Iowa City 52242
Nicola Müller     | 21W. 59th Street    | Westmont, Illinois 60559
```

Il existe un genre de table particulier qui ne peut contenir qu'un seul enregistrement. Ce genre de table est très utile pour contrôler un projet de base de données. Par exemple, vous pouvez y mettre des boutons pour exécuter des tâches diverses ou un champ en lecture seule pour afficher une information concernant le projet.

Voici un exemple : supposez que vous ayez une base de données pour gérer votre compte bancaire dans laquelle vous entrez vos revenus et vos dépenses. Une table à un seul enregistrement pourrait contenir un champ en lecture seule de type Réel qui afficherait la différence.

Chaque table dispose de deux pointeurs d'enregistrement : un vers l'enregistrement actuellement affiché dans l'interface graphique (nommé *pointeur d'enregistrement GUI*) et un vers l'enregistrement courant lors de l'exécution d'un programme BeeBase (nommé *pointeur d'enregistrement de programme*).

Vous pouvez définir autant de tables que vous le désirez dans un projet BeeBase. Les tables peuvent être ajoutées, renommées ou effacées d'un projet.

## 5.3 Enregistrements

Un enregistrement correspond à une ligne d'une table. Il contient toutes les informations d'un élément de la table, par exemple dans une table gérant des adresses, un enregistrement correspond à une adresse.

Chaque enregistrement se voit assigner un numéro qui définit son emplacement dans la table. Ce numéro peut changer si vous ajoutez ou effacez des enregistrements.

Il existe pour chaque table un enregistrement appelé *initial record* avec des valeurs par défaut qui permet l'entrée de nouveaux enregistrements. L'enregistrement initial porte toujours le numéro d'enregistrement 0.

Les enregistrements d'une table peuvent être ajoutés, modifiés ou effacés. Ils ne sont pas forcément présents en mémoire mais sont chargés et stockés quand c'est nécessaire. Le nombre maximum d'enregistrements dans une table est limité par deux choses: Le numéro de l'enregistrement, est une valeur 32 bits de type entier, ce qui donne en théorie un maximum de 4 294 967 295 enregistrements. La deuxième limitation (et la plus contraignante) concerne la mémoire parce que chaque enregistrement génère un petit fichier qui est gardé en mémoire. Malgré tout, BeeBase est capable de travailler sur 10,000 enregistrements et plus.

## 5.4 Champs

Un champ correspond à une colonne d'une table. Il définit le type et l'apparence de la colonne correspondante.

Les champs d'une table peuvent être ajoutés, renommés ou effacés. Le nombre de champ par table n'est pas limité.

Pour chaque champ vous devez définir un type qui spécialise le contenu de ce champ. Reportez-vous au paragraphe suivant pour connaître la liste des types de champs disponibles.

## 5.5 Types de champ

Les champs peuvent être de type Texte, Entier, Réel, Booléen, Choix, Date, Heure, Mémo, Référence, Virtuel, ou Bouton. Ces types sont décrits plus en détail ci-dessous.

Certains types de champ peuvent avoir une valeur spéciale appelée NIL. Cette valeur spéciale se traduit par une valeur non définie. Par exemple pour un type date, elle est interprétée comme une date inconnue. La valeur NIL est similaire à la valeur NULL que l'on trouve dans d'autres systèmes de base de données.

Notez qu'une fois le type de champ défini, il ne peut plus être changé.

### 5.5.1 Champ Texte

Les champs de type texte contiennent une seule ligne de texte (0 à 999 caractères). Les champs Texte sont les types de champ le plus souvent rencontrés dans un projet de base de données. Par exemple, une base de données relationnelle pourrait contenir le nom, la rue et la ville d'une personne, chacun dans son propre champ Texte.

Pour un champ Texte, vous devez définir le nombre maximum de caractères autorisés. Ce nombre n'affecte pas directement la consommation en mémoire ou la place sur le disque parce que seul le contenu effectif du champ est stocké (d'autres systèmes de base de données

appellent cette fonctionnalité texte compressé). En cas de besoin, ce nombre peut être modifié après avoir défini un champ Texte.

Les champs texte peuvent également servir à stocker des polices de caractères et des noms de fichiers. Concernant les noms de fichiers, des afficheurs externes peuvent être appelés pour afficher leur contenu. De plus, une classe image interne permet de visualiser un fichier contenant une image.

Les champs texte ne peuvent pas prendre la valeur spéciale NIL.

### 5.5.2 Champ Entier

Les champs de type Entier stockent des nombres entiers compris entre -2 147 483 648 et 2 147 483 647. Ils sont principalement utilisés pour stocker des quantités de toute nature, par exemple le nombre d'enfants d'une personne, le nombre de titres présents sur un CD.

Les champs Entier peuvent prendre la valeur spéciale NIL, ce qui équivaut à un nombre entier indéfini.

### 5.5.3 Champ Réel

Les champs de type Réel stockent des nombres à virgules compris entre -3,59e308 et +3,59e308. Ils sont utilisés pour stocker des nombres de toute nature, par exemple des sommes d'argent perçues ou dépensées.

Pour chaque champ Réel, vous pouvez définir le nombre de chiffres après la virgule pour l'affichage, bien que la valeur stockée soit d'une précision arbitraire.

Les champs réels peuvent prendre la valeur spéciale NIL, ce qui équivaut à un nombre réel indéfini.

### 5.5.4 Champ Booléen

Les champs de type Booléen stockent une seule information. Ils sont utilisés pour stocker des valeurs oui/non ou vrai/faux, par exemple dans un projet de gestion de commandes, un champ Booléen peut stocker l'information '`paiement effectué ?`'.

Les champs Booléen ne peuvent avoir que TRUE (mot anglais pour VRAI) ou NIL comme valeur. Dans ce cas, NIL équivaut à la valeur FALSE (mot anglais pour FAUX).

### 5.5.5 Champ Choix

Les champs de type Choix enregistrent une donnée parmi une liste de données. Par exemple, pour un projet de carnet d'adresses un champ de type Choix peut être utilisé pour enregistrer le nom du pays, qui se trouve dans une liste de pays du genre '`France`', '`Canada`', '`Allemagne`' ou '`autres`'.

Les champs de type Choix n'enregistrent pas le texte lui-même mais le numéro correspondant au texte dans la liste de données (index). Le nombre de données ainsi que les données peuvent être modifiées une fois que le champ a été créé. Cependant si vous modifiez un champ de type Choix, les valeurs des enregistrements existants ne seront pas modifiées pour refléter ce changement.

Les champs de type Choix ne peuvent pas prendre la valeur NIL.

### 5.5.6 Champs Date

Les champs de type Date enregistrent des dates du calendrier. Par exemple un champ de type Date pourrait stocker des dates d'anniversaire.

Pour visualiser ou enregistrer des champs de type Date, il faut que le format soit du style '`JJ.MM.AAAA`', '`MM/JJ/AAAA`' ou '`AAAA-MM-JJ`', dans lequel '`JJ`', '`MM`' et '`AAAA`' sont des valeurs à 2 et 4 chiffres qui correspondent respectivement au jour, au mois et à l'année.

Les dates peuvent prendre la valeur NIL, ce qui correspond à une date indéfinie.

### 5.5.7 Champ Heure

Les champs de type Heure enregistrent l'heure ou la durée. Par exemple, un champ de type Heure pourrait enregistrer la durée des pistes d'un CD.

Pour visualiser ou enregistrer des champs de type Heure, il faut que le format soit du style '`HH:MM:SS`', '`MM:SS`' ou '`HH:MM`', dans lequel '`HH`' représente les heures, '`MM`' les minutes, et '`SS`' les secondes. En interne, les heures sont stockées en nombre de secondes à partir de 00:00. Les heures peuvent dépasser 23:59:59 jusqu'au temps maximum de 596523:14:07, en revanche les valeurs négatives ne sont pas possibles.

Les heures peuvent prendre la valeur NIL, ce qui signifie une durée non définie.

### 5.5.8 Champ Mémo

Les champs de type Mémo enregistrent du texte de longueur variable et sur plusieurs lignes. La longueur du texte est gérée dynamiquement ce qui veut dire que la mémoire est allouée d'après la longueur actuelle du texte. Par exemple dans un projet de gestion de films, un mémo pourrait enregistrer les résumés des films.

Les mémos ne peuvent pas prendre la valeur NIL.

### 5.5.9 Champ Référence

Les champs de type Référence sont particuliers et n'existent probablement pas dans d'autres systèmes de base de données. Les références stockent un pointeur vers un autre enregistrement. Cet autre enregistrement peut se trouver dans la même table que celle contenant la référence mais aussi dans une toute autre table.

Par exemple dans un projet de gestion généalogique, deux champs de type Référence pourraient stocker les pointeurs vers l'enregistrement respectivement du père et de la mère. Autre exemple dans un projet de gestion de CD et de chansons, une table contenant les chansons pourrait avoir une référence qui pointerait vers les enregistrements des CD correspondants.

Pour l'affichage d'une référence, n'importe quel champ de l'enregistrement référencé peut être utilisé. La valeur d'une référence peut être saisie en sélectionnant un enregistrement parmi la liste des enregistrements de la table référencée.

Les références peuvent prendre la valeur NIL, ce qui correspond à un pointeur vers l'enregistrement initial de la table "externe".

### 5.5.10 Champ Virtuel

Les champs de type Virtuel n'enregistrent pas d'information dans la base de donnée elle-même mais effectuent des calculs dynamiques quand c'est nécessaire.

Par exemple, dans un projet de gestion de factures où un champ Réel contiendrait les sommes hors taxes, un champ Virtuel pourrait calculer les prix TTC. Le champ Virtuel serait calculé à partir des valeurs hors taxes à chaque fois que l'on voudrait connaître sa valeur.

Il existe trois façons de visualiser des champs Virtuel : booléen, texte et liste. Cela permet d'afficher la valeur d'un champ Virtuel sous forme de valeur VRAI/FAUX, de texte sur une seule ligne en incluant nombres, dates et heures, ou sous forme d'une liste de textes. Par exemple pour afficher tous les titres d'un CD.

Les champs Virtuel peuvent prendre la valeur NIL, ce qui correspond à FAUX dans le mode booléen, indéfini dans le mode texte et vide dans le mode liste de textes.

### 5.5.11 Champ Bouton

Les champs de type Bouton ne constituent pas vraiment un type de champ parce qu'ils ne stockent pas et n'affichent pas d'information. Les boutons sont plutôt utilisés pour lancer des programmes BeeBase.

## 5.6 Tableau des types de champ

La table suivante récapitule tous les types champ disponibles :

| Type | Nil? | Description |
|---|---|---|
| Texte | Non | Pour stocker du texte (0 à 999 caractères). Le texte peut servir également à stocker des noms de fichiers, des noms de polices ou une chaîne dans une liste de chaînes. Pour les noms de fichiers, vous pouvez ajouter une zone montrant le contenu du fichier sous forme d'image. |
| Entier | Oui | Pour stocker des nombres entiers (-2 147 483 648 à 2 147 483 647). |
| Réel | Oui | Pour stocker des nombres réels (-3,59e308 à +3,59e308). |
| Booléen | Oui | Valeur booléenne TRUE (VRAI) ou NIL (FAUX). |
| Choix | Non | Un nombre parmi N nombres. Les nombres sont représentés par des entrées de texte. |
| Date | Oui | Pour stocker des dates (01.01.0000 - 31.12.9999). |
| Heure | Oui | Pour stocker des horaires (00:00:00 - 596523:14:07). |
| Mémo | Non | Texte sur plusieurs lignes de longueur illimitée. |
| Référence | Oui | Pour stocker un pointeur vers un enregistrement dans une autre table (NIL = enregistrement initial). |

```
Virtuel    Oui   Pour afficher des résultats à partir d'un programme BeeBase.

Bouton     Non   Pour lancer une fonction programmée.
```

## 5.7 Relations

Maintenant vous êtes en mesure de mettre vos informations sous forme de table avec enregistrements et champs. Mais peut-être voudriez-vous associer différentes tables.

Par exemple, si vous répertoriez votre collection de CD, vous pourriez avoir besoin de deux tables, une pour les CD eux-mêmes et une autre pour les chansons se trouvant sur ces CD. Bien sûr vous pourriez également avoir toutes les chansons dans la table du CD mais dans ce cas, vous auriez un nombre fixe de titres par CD.

Maintenant que vous avez ces deux tables, il est nécessaire de relier chaque chanson au CD dont elle fait partie. Cette liaison entre deux tables est appelée *relation*. Dans BeeBase, on utilise un champ de type Référence pour mettre en place une relation.

Lors de la mise en place d'un champ de type Référence dans une table, vous créez automatiquement une relation entre la table contenant ce champ et la table à laquelle il se réfère.

### 5.7.1 Relations « Un à Un »

Les relations « Un à Un » sont les plus simples. Pour chaque enregistrement on a zéro ou un partenaire dans la même table ou d'une autre table.

Par exemple dans une base de données gérant vos acteurs favoris, vous pourriez utiliser un champ de type Référence nommé '`marie avec`' qui montrerait la personne avec qui cet acteur est marié. Un acteur qui serait célibataire aurait une valeur NIL pour ce champ.

Évidemment, personne ne vous empêche d'associer le champ '`marie avec`' de plusieurs acteurs avec la même personne. Quoi qu'il en soit en programmant BeeBase, il est possible de déceler ce genre de cas et d'agir en conséquence.

### 5.7.2 Relations « Un à plusieurs »

Les relations « Un à plusieurs » sont utiles pour associer un ensemble d'enregistrements avec un enregistrement de la même table ou d'une autre table.

Par exemple, pour un projet de gestion de compte bancaire vous pourriez définir une table pour tous vos comptes bancaires et une autre table répertoriant toutes vos transactions. Maintenant vous voudriez sûrement savoir à quel compte appartient une transaction. Donc vous créez, dans la table des transactions, un champ de type Référence associé à la table des comptes.

Les relations « Un à plusieurs » sont les relations les plus couramment utilisées. Vous pouvez les utiliser pour gérer toute structure de type hiérarchique, par exemple des CD et leurs chansons ou des comptes bancaires et leurs transactions ou encore des arbres généalogiques, etc.

Les relations « Un à plusieurs » servent également de base à la réalisation de relations « Plusieurs à plusieurs » comme vous le verrez dans le chapitre suivant.

### 5.7.3 Relations « Plusieurs à plusieurs »

Les relations « Plusieurs à plusieurs » sont utilisées dans le cas où vous voudriez relier un ensemble d'enregistrements à un autre ensemble d'enregistrements.

Par exemple, dans un projet de gestion d'acteurs et de films, vous auriez deux tables, une pour les acteurs et une autre pour les films. Supposez que vous vouliez connaître les acteurs d'un film. Dans ce cas vous pourriez créer, dans la table des acteurs, un champ Référence vers la table des films. Mais de cette façon, vous ne pourriez avoir qu'un seul film référencé pour chaque acteur parce qu'il n'existe qu'un seul champ Référence dans la table des acteurs. Ce dont vous avez besoin c'est d'un nombre illimité de références depuis la table des acteurs vers la table des films.

Cette opération s'effectue en ajoutant une nouvelle table qui ne contient que deux champs Référence, l'un pointant vers la table des acteurs tandis que l'autre pointera vers la table des films. À partir de maintenant vous pouvez établir des relations en ajoutant de nouveaux enregistrements à cette table. Pour chaque relation acteur-film, vous ajoutez un nouvel enregistrement et sélectionnez l'acteur et le film dans les champs Référence correspondants.

Pour savoir dans quels films a joué un acteur, vous n'avez plus qu'à chercher dans la nouvelle table tous les enregistrements relatifs à cet acteur, ce qui montrera tous les films qui lui sont associés. BeeBase pourrait automatiquement effectuer ce genre de recherche et afficher le résultat dans une liste.

Les tables suivantes montrent comment relier un groupe d'acteurs avec un groupe de films.

```
      Titre            Pays
      -----------------------------
m1:   Batman           USA
m2:   Batman Returns   USA
m3:   Speechless       USA
m4:   Tequila Sunrise  USA
m5:   Mad Max          Australie
m6:   Braveheart       USA
```

```
      Nom
      -------------------
a1:   Michael Keaton
a2:   Jack Nicholson
a3:   Kim Basinger
a4:   Danny DeVito
a5:   Michelle Pfeiffer
a6:   Geena Davis
a7:   Christopher Reeve
a8:   Mel Gibson
a9:   Kurt Russell
a10:  Sophie Marceau
```

```
a11:  Patrick McGoohan
a12:  Catherine McCormack
a13:  Christopher Walken
```

| FilmRef | ActeurRef |
|---|---|
| m1 | a1 |
| m1 | a2 |
| m1 | a3 |
| m2 | a1 |
| m2 | a4 |
| m2 | a5 |
| m2 | a13 |
| m3 | a1 |
| m3 | a6 |
| m3 | a7 |
| m4 | a8 |
| m4 | a5 |
| m4 | a9 |
| m5 | a8 |
| m6 | a8 |
| m6 | a10 |
| m6 | a11 |

À partir de ces tables, vous pouvez voir que Mel Gibson a joué dans les films Tequila Sunrise, Mad Max et Braveheart ou que dans le film Batman, les acteurs principaux sont Michael Keaton, Jack Nicholson et Kim Basinger.

## 5.8 Interface graphique

BeeBase utilise une interface graphique (IHM) organisée de façon hiérarchique pour afficher le contenu des enregistrements et stocker des données. Chaque projet utilise sa propre fenêtre principale, qui contiendra les éléments graphiques ainsi que les fenêtres secondaires. Ces éléments graphiques sont aussi appelés *objets d'affichage*.

Une table est affichée dans son propre environnement visuel appelé *fiche*. Une fiche ne peut afficher qu'un seul enregistrement à la fois. Sa représentation ainsi que les champs qui composent la fiche sont modifiables par l'utilisateur.

Les chapitres suivants détaillent les éléments graphiques dont dispose BeeBase pour créer l'environnement visuel d'un projet.

### 5.8.1 Objet Fenêtre

Les fenêtres sont utilisées pour scinder les informations d'un projet en zones indépendantes.

Chaque projet a automatiquement sa propre fenêtre principale. En cas de besoin, par exemple si la taille de la fenêtre principale est trop grande, on peut créer des fenêtres supplémentaires, qui peuvent elles-mêmes avoir des fenêtres secondaires.

Pour chaque fenêtre secondaire, on peut placer un bouton dans la fenêtre du niveau supérieur afin de faciliter l'ouverture de la fenêtre secondaire. Ce bouton ressemble à un bouton de texte mais peut contenir un petit icône pour le distinguer des autres boutons.

La fenêtre principale ne peut avoir de fenêtre au niveau supérieur. Elle est donc dépourvue de bouton fenêtre. Fermer une fenêtre principale équivaut à fermer le projet en cours.

Une fenêtre peut avoir d'autres éléments graphiques pour enfants. Une image par défaut est affichée si aucun enfant n'a été ajouté à la fenêtre.

### 5.8.2 Objet Fiche

Une fiche permet d'afficher le contenu d'une table. On ne peut voir qu'un enregistrement de la table à la fois.

La fiche peut incorporer un onglet pour contrôler la table (voir le paragraphe suivant). Les autres éléments graphiques tels que les objets Champ ou Texte peuvent être placés dans une fiche pour montrer le contenu d'un enregistrement.

Les fiches ne peuvent pas être placées à l'intérieur d'autres fiches car cela générerait une hiérarchie de fiches et par prolongement de tables. Pour mettre en place une hiérarchie de tables, on utilise une relation de type 1:n entre deux tables.

### 5.8.3 Objet Onglet

Un onglet est une zone rectangulaire de petite taille mais large positionnée contre le bord supérieur d'une fiche. Un onglet peut afficher un titre, par exemple le nom correspondant à la table, une paire de chiffres montrant le numéro de l'enregistrement actuel et le nombre total d'enregistrements, et plusieurs boutons pour contrôler la table, par exemple pour afficher l'enregistrement précédent ou suivant.

On ne peut définir qu'un onglet par fiche. Si vous mettez un onglet dans une fiche, cela se traduit par une bordure supplémentaire, sinon aucune bordure n'est affichée.

### 5.8.4 Objet Champ

Les objets Champ sont utilisés pour afficher le contenu d'un des champs d'un enregistrement.

Suivant le type de champ, l'élément graphique sera soit une zone de texte (pour les types texte, entier, réel, date et heure), une case à cocher (pour le type booléen), un bouton cyclique ou une série de boutons radio (pour le type choix), une zone d'édition (pour le type mémo), une liste (pour le type référence), un texte, une case à cocher ou une liste (pour le type virtuel) et enfin un bouton texte ou un bouton image (pour le type bouton). Dans certains cas, l'élément graphique peut être une zone de texte si l'objet de champ est en lecture seule.

### 5.8.5 Objet Texte

Les objets Texte sont utilisés pour introduire ou décrire les autres éléments d'une fiche ou pour afficher un texte statique.

### 5.8.6 Objet Image

Les images peuvent être affichées n'importe où dans une fenêtre. Une image peut être un motif, une zone colorée ou bien une image d'un fichier externe. La taille de l'image peut être fixe ou redimensionnable.

Une image est statique. Pour stocker des images dans une table, vous devez utiliser un champ de type Texte (voir Section 5.5.1 [String type], page 21).

### 5.8.7 Objet Espacement

Les objets Espacement sont utilisés pour insérer des espaces dans la présentation d'une fenêtre ou dans la fiche d'une table. Un objet Espacement peut avoir une barre horizontale (ou verticale) pour délimiter d'autres éléments graphiques.

### 5.8.8 Objet Groupe

Les éléments graphiques peuvent être classés en groupes horizontaux ou verticaux. Le placement des éléments se fait de gauche à droite (groupe horizontal) ou du haut vers le bas (groupe vertical).

Un groupe peut entourer ses éléments d'un cadre rectangulaire, peut afficher un titre en haut du groupe et peut insérer des espaces entre ses éléments.

### 5.8.9 Objet Balance

Les objets Balance peuvent être placés entre d'autres objets d'une fenêtre, d'une fiche ou d'un groupe. Un objet Balance permet à l'utilisateur de contrôler les dimensions de ses éléments fils et par conséquent la place prise par chacun.

### 5.8.10 Objet Registre

Un registre peut être utilisé pour disposer les éléments graphiques sur plusieurs pages mais avec une seule de ces pages visible à la fois. Cela est très utile lorsque l'interface utilisateur devient trop importante et que vous ne voulez pas l'étendre sur plusieurs fenêtres.

# 6 Gestion des projets

Dans ce chapitre, vous trouverez comment BeeBase organise des projets, comment les ouvrir, les sauvegarder, les supprimer et les fermer. Comment vérifier l'intégrité des données et décharger les enregistrements.

## 6.1 Format de fichier

Un projet BeeBase est stocké dans une base SQLite3. Si vous êtes curieux, vous pouvez utiliser une visionneuse ou un éditeur de fichier SQLite3, comme '`DB Browser for SQLite`', pour explorer la structure interne d'un projet BeeBase. Le contenu de la base SQLite3 est compréhensible car il est dans un format lisible. Pour autant certaines choses ne sont pas évidentes.

- Les tables et champs SQLite commençant par un caractère '`_`' sont utilisées par BeeBase pour son fonctionnement interne : journalisation, définition de la structure et de la mise en page du projet ;
- Les autres tables sont celles définies par l'utilisateur du projet BeeBase ;
- Les dates sont des entiers représentant le nombre de jours écoulés depuis le jour '`01/01/0000`' du '`calendrier grégorien`' ;
- Les heures sont des entiers représentant le nombre de secondes écoulées depuis '`00:00:00`' ;
- Les références d'enregistrement sont représentés par un entier qui correspond à la valeur du champ '`_ID`' de l'enregistrement référencé ;
- Tous les IDs d'enregistrement (stockés dans '`_ID`') sont recalculés lorsque qu'un projet est réorganisé.

Il est préférable de ne pas modifier à la main un fichier de projet, mais pour ceux qui veulent expérimenter voici quelques règles à suivre.

- Toute modification devrait être effectuée dans une unique transaction. Par exemple, '`DB Browser for SQLite`' permet de modifier un fichier SQLite et d'écrire toutes les modifications en une transaction ;
- Vous pouvez ajouter, modifier, ou supprimer des enregistrements des tables utilisateur. Lors de l'ajout d'un enregistrement, le champ '`_ID`' doit recevoir une valeur inutilisée, de préférence plus grande que les valeurs de '`_ID`' existantes dans la table. Lors d'une modification d'enregistrement, modifiez aussi le champ '`_Version`' de cet enregistrement (p. ex. incrémenter de 1). Pour que BeeBase remarque les changements du projet, le champ '`Version`' de la table '`_Project`' doit être incrémenté. BeeBase rechargera le projet et ce mettra à jour en tenant compte des enregistrements modifiés ;
- Le champ '`SavedWith`' de la table '`_Project`' devrait être mis à jour avec une explication (p. ex. le nom du programme auteur de la dernière modification du projet) ;
- Modifier les autres tables de BeeBase (celles commençant par '`_`') n'est probablement pas une bonne idée. Mais si vous le faîtes, vous devez aussi incrémenter '`StructureVersion`' en plus de '`Version`' dans la table '`_Project`'.

SQLite permet d'accéder à un projet avec plusieurs instances de BeeBase mais une seule peut modifier le projet. Notez qu'un bon fonctionnement du mécanisme de verrouillage nécessite que le système de fichier offre certaines garanties. Il est recommandé de tester d'abord le partage d'un projet avec plusieurs instances de BeeBase sur un système de fichier particulier avant d'utiliser cette fonctionnalité.

## 6.2 Informations

BeeBase conserve quelques informations sur chaque projet. Sélectionnez le menu '`Projet - Information...`' pour obtenir des informations sur le projet en cours. L'information retournée comprend le nom du projet, le nombre de tables, le nombre d'enregistrements dans toutes les tables, et une valeur montrant le nombre d'octets que vous pourriez gagner si vous réorganisiez ce projet. Le gain n'est cependant qu'une estimation grossière et ne doit pas être considéré comme une valeur exacte. Cette valeur est loin d'être précise, particulièrement si vous avez fait beaucoup de changements sur la structure du projet (en ajoutant ou en enlevant des champs).

## 6.3 Nouveau projet

BeeBase peut manipuler plusieurs projets en même temps. Vous êtes seulement limités par la mémoire disponible. Pour démarrer un autre projet, choisir le menu '`Projet - Nouveau`'. Une nouvelle fenêtre s'ouvre avec un projet vide. Vous pouvez maintenant définir la structure de ce projet (voir Chapitre 15 [Structure editor], page 65) ou charger un projet existant à partir du disque dur (voir Section 6.5 [Open project], page 31).

## 6.4 Nettoyer un projet

Pour réinitialiser un projet, sélectionnez le menu '`Projet - Nettoyer - Projet`'. Le projet en cours sera alors fermé et remplacé par un projet vide. Lorsque vous démarrez BeeBase sans projets vous arrivez automatiquement dans cet état.

Par la sélection du menu '`Projet - Nettoyer - Enregistrements`' vous démarrez un nouveau projet en utilisant la structure courante. Ce qui signifie que tout (excepté les données enregistrées du projet en cours) est employé pour le nouveau projet.

Si un projet en cours n'a pas été sauvé sur le disque dur et que vous avez sélectionné le menu ci-dessus, une fenêtre vous demandera alors la confirmation de l'opération.

## 6.5 Ouvrir un projet

Pour charger un projet sélectionnez le menu '`Projet - Ouvrir - Projet`' qui ouvrira une fenêtre vous permettant de choisir un projet. Il y a plusieurs projets de démonstration pour illustrer les fonctionnalités de BeeBase. Pour les charger, sélectionnez le menu '`Projet - Ouvrir - Démo`'.

Si le projet chargé a son programme source placé à l'extérieur, le fichier source externe sera créé après ouverture du projet (voir Section 16.2 [External program source], page 81).

Si vous chargez un projet ou la structure d'un projet alors que le projet courant n'a pas été sauvé, une fenêtre s'ouvrira vous demandant de confirmer l'opération.

## 6.6 Enregistrer le projet

Tous les changements effectués sur un projet se font uniquement en mémoire ou sont temporairement stockés lors de déchargements d'enregistrements (voir Section 6.8 [Swap records], page 33). Aussi, si vous voulez les stocker de façon permanente, vous devez enregistrer le projet sur votre disque dur, en choisissant le menu '`Enregistrer`'. Si votre projet n'a pas encore de nom, il vous sera demandé auparavant.

La raison pour laquelle BeeBase ne sauve pas automatiquement un projet qui a changé, est que, de cette façon, c'est vous qui décidez lorsque vous voulez sauver un projet ! Vous pouvez toujours retourner sur la dernière version sauvée de votre projet en choisissant le menu '`Projet - Rétablir la sauvegarde`'. Ce mécanisme est semblable aux commandes '`COMMIT`' et '`ROLLBACK`' des systèmes de bases de données SQL.

Si vous enregistrez un projet, tous les enregistrements modifiés sont écrits sur le disque dur et le fichier `Structure.bbs` est recréé. Avant de créer le nouveau fichier `Structure.bbs`, BeeBase renomme d'abord le fichier existant `Structure.bbs` en '`Structure.old`' pour avoir une copie de sécurité au cas où l'opération échouerait.

Ce mécanisme garantit le chargement et la sauvegarde rapide des opérations, mais il n'est pas exempté de réorganisation. Si vous avez modifié beaucoup d'enregistrements alors l'emplacement physique où les enregistrements se trouvent et la fragmentation qui en résulte peut devenir désavantageuse. C'est pourquoi un menu '`Projet - Enregistrer & réorganiser`' existe, pour réaliser l'opération d'enregistrement & de réorganisation. Cette opération peut prendre un certain temps selon le nombre et la taille de vos enregistrements. L'opération Enregistrer & Réorganiser crée un nouveau répertoire et réécrit tous les fichiers reliés à un projet. Une fois cette opération réalisée, l'ancien répertoire est supprimé.

Un autre bon moment pour prévoir une réorganisation est lorsque vous avez fait des changements sur la structure de données d'un projet, par exemple après avoir ajouté un nouveau champ dans une table. Ces changements ne sont pas appliqués immédiatement à tous les enregistrements parce que cela prendrait trop de temps de charger chaque enregistrement, pour le modifier, et le sauver de nouveau sur le disque. Par conséquent ces changements sont mis sur une liste interne '`À faire`' qui est appliquée après le chargement d'un enregistrement. L'application de cette liste à un enregistrement prend peu de temps. Cependant plus la liste devient longue, plus cela prend de temps. Réorganiser un projet oblige à appliquer cette liste '`à faire`' à tous les enregistrements, donc si vous avez fait des modifications à la structure du projet alors sa réorganisation réduira le temps de chargement des enregistrements.

Vous pouvez également enregistrer et réorganiser un projet sous un nouveau nom, ce qui conserve l'ancien projet intact. Pour faire cela il suffit d'utiliser le menu '`Projet - Enregistrer & réorganiser sous`' qui vous demande de saisir un nouveau nom de projet.

## 6.7 Modes Administrateur et Utilisateur

BeeBase travaille soit en mode Administrateur (mode par défaut), soit en mode utilisateur. Vous pouvez basculer d'un mode à l'autre en sélectionnant le menu '`Projet - Passer en mode Administrateur`' et '`Projet - Passer en mode Utilisateur`'. En mode Utilisateur, plusieurs éléments de menu sont désactivés et les éditeurs de structure, de programme et de requêtes ne sont pas disponibles. Ainsi seule l'édition basique d'enregistrement est possible. En mode Administrateur toutes les opérations sont autorisées.

Un mot de passe Administrateur peut être configuré pour un projet en sélectionnant le menu '`Projet - Modifier le mot de passe Administrateur`'. Une fois configuré, le mot de passe doit être saisi pour passer en mode Administrateur sinon le basculement sera refusé laissant le projet en mode Utilisateur.

Lors de l'ouverture d'un projet ayant un mot de passe Administrateur configuré, le projet est démarré en mode Utilisateur, dans le cas contraire (aucun mot de passe Administrateur n'est configuré), il est démarré en mode Administrateur.

## 6.8 Décharger les enregistrements

BeeBase n'a pas besoin de conserver tous les enregistrements d'un projet en mémoire. Ainsi le chargement et la sauvegarde des projets sont beaucoup plus rapides. En chargeant un projet, un entête d'enregistrement est alloué pour chaque enregistrement. Les données elles-mêmes sont chargées seulement lorsque c'est nécessaire, par exemple quand elles sont affichées sur l'écran. Le nombre d'enregistrements est encore limité par la mémoire disponible puisque chaque entête d'enregistrement utilise quelques octets de mémoire.

Vous pouvez spécifier la taille de la mémoire que devrait utiliser BeeBase pour les enregistrements d'un projet. Choisir une des valeurs prédéfinies se trouvant dans le menu '`Préférences - Cache d'enregistrements`' (voir Section 7.2.1 [Record memory], page 36). BeeBase ne préalloue pas un bloc mémoire de la taille indiquée, il vérifie seulement de temps en temps si la quantité courante de mémoire allouée est plus grande que la valeur indiquée.

Si BeeBase manque de mémoire ou si la limite supérieure de la capacité mémoire a été atteinte alors BeeBase essaye de libérer autant d'enregistrements de la mémoire que possible. Dans ce cas-là, BeeBase peut écrire les enregistrements modifiés sur le disque dur pour obtenir le maximum de mémoire disponible. Vous pouvez également forcer BeeBase à le faire en choisissant le menu '`Projet - Décharger les enregistrements`'.

BeeBase maintient une liste d'espace libre pour chaque fichier d'enregistrements. Si vous supprimez un enregistrement, l'emplacement dans le fichier d'enregistrements est ajouté à la liste d'espace libre. En outre, si vous changez un enregistrement et que celui-ci a besoin d'être écrit sur le disque dur, l'ancien emplacement dans le fichier est ajouté à la liste d'espace libre. Cependant BeeBase s'assure qu'un rechargement vous permettra toujours de retourner au point de la dernière opération de sauvegarde. BeeBase n'écrira pas sur les secteurs qui sont libérés mais où un enregistrement existe toujours et qui pourra être accédé en ouvrant à nouveau le projet.

## 6.9 Fermer le Projet

Quand vous avez terminé votre travail sur un projet vous pouvez le fermer en sélectionnant le menu '`Projet - Fermer`'. Ce qui libère la mémoire et toutes les ressources appartenant au projet. Si le projet contient des changements qui n'ont pas encore été sauvés, une fenêtre s'ouvrira vous demandant de sauver, de continuer ou d'annuler l'opération.

Pour la fermeture d'un projet vous pouvez également choisir le menu '`Projet - Sauver & Fermer`' qui sauve d'abord le projet s'il y a eu des changements avant de le fermer.

# 7 Préférences

BeeBase offre à l'utilisateur un certain nombre d'éléments de préférence qu'il peut configurer selon ses goûts. Ce chapitre présente quelles options sont disponibles et donne des informations générales sur la façon dont le système de préférence fonctionne.

Les options sont divisées en « réglages utilisateur » et « réglages projet ».

## 7.1 Réglages utilisateur

Les réglages utilisateur comprennent les éléments de configuration dépendant de l'utilisateur, p. ex. la langue de l'utilisateur, son pays ou ses goûts Les réglages utilisateur sont affichés et peuvent être configurés en haut du menu '`Préférences`'.

Les réglages utilisateur sont stockés dans l'environnement de l'utilisateur. Sur Windows, Mac OS et Linux ils sont conservés dans le répertoire personnel de l'utilisateur dans `.BeeBase.prefs`. Sur Amiga ils sont stockés dans `ENV:BeeBase.prefs` et `ENVARC:BeeBase.prefs`.

Les réglages utilisateur contiennent les éléments de configuration listés ci-dessous. À chaque fois que l'un de ces éléments est modifié dans le menu '`Préférences`', les réglages sont sauvegardés sur disque afin de les rendre permanents.

### 7.1.1 Formats

En sélectionnant le menu '`Préférences - Formats`' vous pouvez spécifier les formats utilisés pour l'affichage et l'impression des valeurs décimales et des dates. La sélection de ce menu ouvre une nouvelle fenêtre contenant les éléments suivants :

- un champ '`Format des nombres réels`' pour positionner le caractère marquant le début de la partie décimale. Vous pouvez choisir entre '`Point décimal`' et '`Virgule décimale`'.
- un champ '`Format des dates`' pour spécifier comment les dates sont affichées. Vous avez le choix entre '`Jour.Mois.Année`', '`Mois/Jour/Année`' et '`Année-Mois-Jour`'.
- deux boutons '`Ok`' et '`Annuler`' pour quitter la fenêtre.

Les valeurs initiales pour les formats des nombres décimaux et des dates sont déterminées en fonction des informations fournies par l'environnement de localisation ('`locale`') du système d'exploitation.

Lorsque vous avez terminé, pressez le bouton '`Ok`' pour quitter la fenêtre et mettre à jour l'affichage.

### 7.1.2 Éditeur externe

En plus de fournir un éditeur intégré à l'interface graphique, BeeBase propose également de modifier les contenus textuels dans un éditeur externe (p. ex. le menu contextuel de l'éditeur interne propose un élément permettant d'appeler l'éditeur externe pour modifier le contenu). Le nom de l'éditeur ainsi que ses paramètres sont spécifiés en choisissant le menu '`Préférences - Éditeur externe`'. Vous devez saisir la ligne de commande à exécuter lors de l'appel de l'éditeur externe. La chaîne spéciale '`%f`' peut être utilisée pour symboliser le nom du fichier et est remplacée avant l'exécution par le nom réel du fichier (ou un nom de fichier temporaire que BeeBase crée pour échanger les données textuelles).

Par exemple, sur Linux, vous pouvez spécifier '`emacs %f`' pour utiliser le puissant éditeur GNU Emacs ; sur Amiga, vous pouvez utiliser '`CED %f -keepio`' pour basculer vers le célèbre éditeur CED comme éditeur externe.

Par défaut la valeur est '`Notepad %f`' sur Windows, '`open -tWn %f`' sur Mac OS, '`gvim -f %f`' sur Linux, et '`Ed %f`' sur Amiga.

Notez que sur Mac OS, la configuration par défaut lance l'éditeur externe de manière synchrone, c.-à-d. que BeeBase attend que l'éditeur se ferme (voir les options '`-W`' et '`-n`' de '`open --help`'). Ce comportement est utilisé quand le menu '`Éditeur externe`' du menu contextuel d'un champ Mémo est lancé (voir Section 9.3 [Changing records], page 45) ou que la commande `EDIT` du langage de programmation est exécutée.

Sur Amiga, la séquence additionnelle '`%p`' est utilisée en la ligne de commande. Lors du lancement de l'éditeur externe, cette séquence est remplacée par le nom de l'écran publique sur lequel BeeBase est affiché.

### 7.1.3 Visionneuse externe

BeeBase utilise une visionneuse externe pour afficher le contenu des fichiers externes, p. ex. les images, les films et toute sorte de documents. Par exemple vous pouvez utiliser le type de données texte pour stocker des noms de fichier dans une table puis laisser BeeBase les afficher en utilisant la visionneuse externe. Pour spécifier cette visionneuse, utilisez le menu '`Préférences - Visionneuse externe`'. Comme pour l'éditeur externe (voir Section 7.1.2 [External editor], page 34), vous pouvez alors saisir une commande.

La valeur par défaut est '`%f`' sur Windows (avec ShellExecute), '`open %f`' sur Mac OS, '`gnome-open %f`' sur Linux, '`Open %f`' sur MorphOS, et '`Multiview %f`' sur les autres systèmes Amiga.

Notez que sur Mac OS, la configuration par défaut lance la visionneuse externe de manière asynchrone, c.-à-d. que BeeBase n'attend pas qu'elle soit fermer. Si vous utilisez Mac OS 10.5 ou plus, vous pouvez utiliser l'option '`open -Wn %f`' pour activer le mode synchrone. Ce comportement est utilisé quand la commande `VIEW` du langage de programmation est exécutée.

Sur Amiga, la séquence additionnelle '`%p`' est utilisée en ligne de commande. Lors du lancement de la visionneuse externe, cette séquence est remplacée par le nom de l'écran publique sur lequel BeeBase est affiché.

À partir de la version 4.0 BeeBase utilise la visionneuse externe par défaut configurée pour votre système d'exploitation et ce paramètre n'est plus disponible.

### 7.1.4 Boutons de navigation cyclique

Dans l'interface graphique spécifiée dans l'éditeur de structure, il est possible qu'il existe un certain nombre de boutons spéciaux. Par boutons spéciaux s'entendent les boutons déroulants, p. ex. les boutons de sélection de fichier ou de polices de caractères attachés à un champ Texte, les boutons d'affichage automatique et de filtrage d'un champ Référence, ainsi que les boutons de visualisation à côté d'un champ texte.

Ces boutons ne sont généralement pas inclus dans le cycle de tabulation, c.-à-d. qu'il est impossible de les activer avec la touche *`Tab`*. Cependant en sélectionnant le menu '`Préférences - Boutons du cycle de tabulation`' ces boutons seront tout de même inclus dans le cycle de tabulation. Cette option est activée par défaut.

Sur Amiga, notez que la modification de l'état de cette option n'a d'effet qu'après reconstruction de l'interface graphique, c'est à dire en basculant vers l'éditeur de structure puis en revenant dans l'interface utilisateur.

### 7.1.5 Champ suivant via <Entrée>

Lorsque le curseur est dans un champ texte simple éditable dans l'interface utilisateur d'un projet, presser la touche ***Entrée*** peut soit passer au prochain élément de l'interface utilisateur, soit laisser le curseur dans le champ texte courant.

En cochant l'élément de menu '`Préférences - Champ suivant via <Entrée>`' le curseur passera au prochain élément de l'interface utilisateur, dans le cas contraire il restera au champ texte courant. Par défaut ce menu est coché.

Veuillez noter que dans tous les cas, presser ***Entrée*** confirme et mémorise le texte saisi.

### 7.1.6 Confirmer la sortie

Si vous essayez de quitter BeeBase et qu'il existe des projets non sauvés, une requête de sécurité est ouverte pour vous demander de confirmer. En revanche si tous les projets sont sauvés, BeeBase se ferme en silence.

Afin de toujours forcer l'affichage d'une requête lors de la sortie de BeeBase, le menu '`Préférences - Confirmer la sortie`' doit être coché. Dans ce cas il est possible de toujours obtenir une demande de confirmation de sortie lors de la sélection du menu '`Projet - Quitter`' ou lorsque le dernier projet ouvert est fermé. Par défaut cette option est décochée.

### 7.1.7 MUI

Sur Amiga uniquement. Comme BeeBase pour Amiga est une application MUI, vous pouvez également indiquer vos préférences MUI pour cette application en utilisant le menu '`Préférences - MUI`'.

## 7.2 Réglages dépendants du projet

Les réglages projet contiennent les éléments de configuration stockés avec le projet. Ces éléments sont affichés et modifiables depuis la partie inférieure du menu '`Préférences`' et dans le menu '`Programme`' (ces deux menus sont séparés pour la clarté et la concision).

Les éléments de configuration suivants appartiennent aux réglages projet. À chaque fois que l'un de ces éléments est modifié, le compteur de modification du projet est incrémenté.

### 7.2.1 Cache d'enregistrements

BeeBase n'a pas besoin de conserver tous les enregistrements d'un projet en mémoire. À la place il utilise un cache pour maintenir en mémoire un petit nombre d'enregistrements. En choisissant une valeur dans le menu '`Préférences - Cache d'enregistrements`' il est possible d'indiquer la taille de ce cache. Chaque projet possède son propre cache, donc si vous ouvrez deux projets chacun avec un cache de 1 Mo, BeeBase utilisera jusqu'à 2 Mo au total.

BeeBase n'alloue pas cette mémoire *a priori*, mais utilise un schéma d'allocation dynamique. De plus la taille spécifiée du cache n'est qu'une limite logicielle et occasionnellement, BeeBase peut être amené à la dépasser.

Lorsque le cache est plein, ou si BeeBase vient à manquer de mémoire, tous les enregistrements sont déchargés du cache. Cela signifie que les enregistrements non modifiés sont simplement libérés tandis que ceux qui ont été modifiés sont d'abord écrits sur le disque puis libérés (voir Section 6.8 [Swap records], page 33).

En donnant à BeeBase une plus grande valeur pour le cache d'enregistrements, vous pourrez noter une augmentation de la rapidité d'accès aux enregistrements car un plus grand nombre d'enregistrements pourront tenir en mémoire ce qui diminue le nombre d'accès disque nécessaires. S'il y a suffisamment de mémoire pour contenir la totalité des enregistrements et que vous avez indiqué une limite de cache suffisante (par exemple '`illimité`') alors BeeBase travaille en vitesse optimale.

Par défaut, la valeur est '`illimité`'.

### 7.2.2 Confirmer la suppression d'enregistrement

Vous devez cocher l'élément de menu '`Préférences - Confirmer la suppression d'enregistrement`' si vous souhaitez que BeeBase demande une confirmation à chaque fois que vous essayez de supprimer un enregistrement. Laissez l'élément décoché si les enregistrements doivent être supprimés silencieusement.

Par défaut cette option est activée.

### 7.2.3 Chemins relatifs au projet

Si vous voulez référencer des données externes (p. ex. des documents ou des images) alors les noms de ces fichiers de données doivent être stockés dans le projet. Pour cela vous pouvez utiliser des chemins absolus ou des chemins contenant des assignations (voir Section 3.8 [Filename conventions], page 8). Un autre moyen est de stocker les données externes relativement au répertoire du projet (note : ce n'est pas le répertoire du projet lui-même mais le répertoire contenant le projet).

En activant le menu '`Préférences - Chemins relatifs au répertoire du projet`', BeeBase change le répertoire de travail pour celui contenant le projet. Cela signifie que si vous avez plusieurs projets ouverts, BeeBase change de répertoire en fonction du projet actif. Une fois que cet élément de menu a été activé pour un projet, les noms de fichiers peuvent être donnés relativement au répertoire du projet. Cela rend un projet indépendant de l'endroit où il est stocké sur le système de fichiers.

Si vous laissez ce menu décoché, BeeBase utilise le répertoire de travail actif au moment où le programme a été lancé.

Par défaut cette option est désactivée.

### 7.2.4 Confirmer le rafraîchissement automatique

Lorsque BeeBase trouve une version à jour de votre projet sur le disque il le recharge automatiquement. Si le menu '`Préférences - Confirmer le rafraîchissement automatique`' est coché BeeBase demande confirmation avant le rechargement. S'il est décoché, le projet est rechargé silencieusement.

Par défaut cette option est activée.

### 7.2.5 Confirmer enregistrement et réorganisation

L'opération d'enregistrement et réorganisation d'un projet peut prendre un certain temps en fonction de la taille du projet. C'est pour cela que lors de la sélection des menus '`Projet - Enregistrer & réorganiser`' ou '`Projet - Enregistrer & réorganiser sous`', il y a demande de confirmation de l'opération. Cette confirmation n'apparaît que si le menu '`Préférences - Confirmer enregistrement et réorganisation`' est coché, il est donc possible de la désactiver en décochant ce menu.

Par défaut cette option est activée.

### 7.2.6 Vacuum après la réorganisation

Si '`Préférences - Vacuum après la réorganisation`' est actif, le fichier de projet SQLite3 est compacté avec la commande '`Vacuum`'. Sa taille est réduite au minimum.

Sur Amiga, cette commande peut échouée quand plusieurs instances de BeeBase accèdent à la même base car la troncature de fichier est limitée. Le fichier ne sera pas corrompu mais il ne sera pas aussi compact qu'il aurait pu l'être.

Par défaut cette option est activée.

### 7.2.7 Code source du programme

En utilisant le menu '`Programme - Code Source`', vous pouvez définir si le code source du programme d'un projet est '`Interne`' ou '`Externe`'. Lorsqu'il est défini comme '`Interne`', il est possible d'utiliser l'éditeur intégré de BeeBase pour modifier et compiler le programme du projet. C'est la valeur par défaut. Si vous souhaitez utiliser un éditeur externe pour programmer, choisissez '`Externe`' et saisissez le nom du nouveau fichier dans lequel BeeBase stockera le code source du programme. Cela vous permet de charger et modifier le code source du programme dans votre éditeur favori. Pour plus d'informations sur la façon d'utiliser cette fonctionnalité voir Section 16.2 [External program source], page 81.

Notez qu'en basculant de '`Externe`' à '`Interne`', la dernière compilation réussie du programme est conservée.

### 7.2.8 Nettoyer les sources des programmes externes

Le menu '`Programme - Nettoyer les sources des programmes externes`' indique si les fichiers sources externes des programmes doivent être effacés lors de la fermeture du projet ou de la modification du menu '`Programme - Code Source`' vers '`Interne`'. Voir Section 16.2 [External program source], page 81, pour plus d'informations à propos des codes sources externes.

Par défaut cette option est activée.

### 7.2.9 Informations de débogage

Lors de la compilation d'un programme du projet, il est possible de choisir si les informations de débogage doivent être incluses dans l'exécutable ou non. Si vous compilez sans ces informations et qu'une erreur survient à l'exécution, une description d'erreur est générée, mais il n'y a pas d'indication sur l'endroit exact où est survenue cette erreur. Si vous compilez avec ces informations, vous obtenez en plus l'endroit exact de l'erreur. Compiler avec les informations de débogage est un peu plus long, nécessite plus de mémoire et le programme résultant est légèrement moins efficace.

Utilisez le menu '`Programme - Inclure information de débogage`' pour activer ou désactiver les informations de débogage. Lorsque vous changez cette option, n'oubliez pas de recompiler le programme du projet en sélectionnant le menu '`Programme - Compiler`'.

Par défaut cette option est activée.

### 7.2.10 Fonctions obsolètes

Depuis la version 2.7 de BeeBase quelques fonctions de programmation sont obsolètes (voir Section 16.30 [List of obsolete functions], page 166). Ces fonctions obsolètes ont été remplacées par d'autres mécanismes et elles ne fonctionnent plus comme attendu. Il est possible de choisir comment gérer le cas lors de l'ouverture de projets contenant des fonctions de programmation devenues obsolètes.

Le menu '`Programme - Fonctions obsolètes`' détermine comment réagir lorsqu'une fonction obsolète est appelée. En choisissant '`Ignorer silencieusement`' tout appel à une fonction obsolète sera ignoré et l'exécution du programme continue en sautant l'appel à la fonction. '`Avertir à l'appel`' ouvre une boite de dialogue informant l'utilisateur qu'une fonction obsolète est appelée et lui permet de poursuivre l'exécution ou de montrer l'emplacement de l'appel dans l'éditeur de programme. C'est l'action par défaut. Si '`Traiter comme une erreur`' est sélectionné, tout appel à une fonction obsolète génère une erreur et la compilation de programmes contenant des fonctions obsolètes échoue avec un message d'erreur approprié.

Il est recommandé de choisir '`Traiter comme une erreur`' après que tous les appels aux fonctions obsolètes ont été supprimés du programme d'un projet. Afin de trouver si votre projet contient des appels à des fonctions obsolètes choisissez '`Traiter comme une erreur`' et recompilez le programme du projet.

### 7.2.11 Trier les déclencheurs

En cochant le menu '`Préférences - Trier les déclencheurs`', les fonctions disponibles en tant que déclencheurs sont triées par ordre alphabétique lors de leur affichage dans les boîtes de dialogue de création et de modification de table (voir Section 15.1.1 [Creating tables], page 65) et dans celles de création et modification de champ (voir Section 15.2.1 [Creating fields], page 67). Dans le cas contraire, les fonctions sont affichées dans leur ordre d'apparition dans le programme du projet.

Par défaut cette option est désactivée.

### 7.2.12 Répertoire d'inclusion

La fonction de programmation de BeeBase autorise l'inclusion dans le programme du projet de fichiers sources externes (voir Section 16.3.3 [#include], page 83, pour plus de détails). Le menu '`Programme - Répertoire d'inclusion`' permet de spécifier le répertoire où BeeBase doit rechercher ces fichiers d'inclusion.

La valeur par défaut est '`BeeBase:Include`'.

### 7.2.13 Fichier de sortie du programme

Lors de l'exécution d'un programme BeeBase, toutes les sorties à destination de la sortie standard ('`stdout`') sont redirigées vers un fichier. Le nom de ce fichier peut être saisi dans une boite de dialogue ouverte lorsque l'on sélectionne le menu '`Program - Fichier de`

`sortie du programme`'. Vous pouvez également spécifier si les sorties doivent être ajoutées en fin de fichier ou si le fichier doit être réinitialisé (i.e. effacé puis recréé) à la première sortie.

Sur Windows, Mac OS et Linux, vous pouvez diriger la sortie vers un programme externe pour en traiter les données. Pour cela, le premier caractère du nom de fichier doit être le symbole pipe '`|`' suivi du nom du programme externe et de ses arguments. Le programme externe doit lire les données à partir de son entrée standard ('`stdin`'). Par exemple, cela permet sous Linux les redirections suivantes :

- `|lpr` imprime sur l'imprimante par défaut.
- `|mknod /tmp/pipe p; (xterm -e less -f /tmp/pipe &); cat > /tmp/pipe; rm /tmp/pipe` affiche en utilisant le programme '`less`' dans sa propre fenêtre '`xterm`'.
- `/dev/pts/0` redirige sur le terminal connecté sur '`pts/0`'. Sur certains bureaux, p. ex. KDE, un démon est à l'écoute de ce terminal et la sortie est redirigée dans une fenêtre.
- `/dev/stdout` redirige sur la sortie standard de BeeBase. C'est la valeur par défaut.

Sur Amiga, il existe quelques fichiers spéciaux pour rediriger la sortie, p. ex. :

- `PRT:` imprime sur l'imprimante.
- `CON:////BeeBase output/CLOSE/WAIT` affiche dans une fenêtre Shell.
- `CONSOLE:` redirige vers la fenêtre Shell depuis laquelle BeeBase a été lancé. C'est la valeur par défaut.

## 7.3 Enregistrer des réglages par défaut

Les réglages projet sont spécifiques à chaque projet, c.-à-d. des projets différents peuvent avoir des réglages différents. Pour les nouveaux projets des réglages par défaut sont choisis. Ces réglages par défaut sont stockés avec les réglages utilisateurs dans l'environnement de l'utilisateur (voir Section 7.1 [User settings], page 34).

Afin de personnaliser les réglages par défaut que vous souhaitez, vous pouvez stocker les réglages du projet courant comme étant les réglages par défaut des nouveaux projets en sélectionnant le menu '`Préférences - Enregistrer comme défaut`'.

# 8 Journal

Ce chapitre décrit la fonctionnalité de journalisation de BeeBase : la table interne '_Log', comment activer la journalisation, son mode et le fichier journal externe ; la définition d'un tag pour les nouvelles entrées, comment importer, exporter, vider un journal, et appliquer des modifications provenant d'un journal, comment afficher toutes les entrées du journal.

## 8.1 Table Log

Pour journaliser les modifications d'un projet, BeeBase utilise une table interne nommée '_Log'. Notez que le souligné en début de nom indique que cette table est une table système gérée par BeeBase qui ne peut pas être modifiée.

Cette table contient les champs suivants :

| Nom | Description |
|---|---|
| '_Label' | Un champ qui peut être défini via le menu 'Journal - Définir le tag' ou calculé par une fonction déclencheur nommée logLabel (voir Section 8.5 [Set log label], page 42). |
| '_Date' | Date de l'entrée. |
| '_Time' | Heure de l'entrée. |
| '_Action' | 'Nouvel enregistrement', 'Enregistrement supprimé' Ou 'Valeur modifiée'. |
| '_Table' | Nom de la table sur laquelle l'action a eu lieu. |
| '_Record' | Numéro d'enregistrement concerné par l'action. Notez que le numéro d'enregistrement peut changer après une compression ou tri des enregistrements d'une table. |
| '_Description' | Description de l'enregistrement concerné par l'action. La description de l'enregistrement peut être spécifiée dans l'éditeur (voir Section 15.3.2 [Table object editor], page 71). |
| '_Field' | Si l'action est une 'Valeur modifiée' alors le champ porte le nom du champ ayant été modifié. |
| '_OldValue' | Ancienne valeur du champ (pour l'action 'Valeur modifiée'). |
| '_NewValue' | Nouvelle valeur du champ (pour l'action 'Valeur modifiée'). |

La table '_Log' peut être lue comme les autres tables d'un projet. Par exemple, une requête select-from-where

```
SELECT * from _Log
```

peut être exécutée dans l'éditeur de requête pour obtenir la liste de toutes les modifications (voir Section 14.1 [Select-from-where queries], page 59, et Section 14.2 [Query editor], page 59).

## 8.2 Activer la journalisation

Par défaut la journalisation est désactivée dans un projet. Pour commencer à enregistrer les modifications sélectionnez le menu 'Journal - Activer la journalisation'. L'état de

ce menu est enregistré dans le fichier du projet, ainsi une fois activé et enregistré les modifications continueront à s'enregistrer même à l'ouverture suivante du projet. Désactiver le menu pour arrêter la journalisation.

Notez que seules les modifications des tables et champs ayant l'option '`Journaliser les modifications`' seront enregistrées (voir Section 15.1.1 [Creating tables], page 65, et Section 15.2.1 [Creating fields], page 67).

Les modifications sons journalisées lors de l'ajout, la suppression ou la modification d'un champ, d'un enregistrement. Pour chacune de ces actions une nouvelle entrée est crée et ajoutée à la table '`_Log`' si le mode de journalisation inclut la journalisation dans le projet (voir Section 8.3 [Logging mode], page 42).

## 8.3 Mode de journalisation

Vous pouvez sélectionner un de ces modes de journalisation dans le menu '`Journal - Mode de journalisation`' :

- '`Dans le projet`' : les modifications sont journalisées dans la table '`_Log`' du projet (choix par défaut) ;
- '`Dans un fichier`' : les modifications sont journalisées vers un fichier externe ;
- '`Dans les deux`' : les modifications sont journalisées dans la table '`_Log`' du projet et vers un fichier externe.

Les modes qui incluent la journalisation dans un fichier externe sont utiles pour préserver les modifications si le fichier de projet vient à être corrompu. Voir Section 8.9 [Apply changes], page 43, pour savoir comment l'utiliser pour restaurer un projet à partir d'une sauvegarde.

## 8.4 Définir le fichier journal

Pour définir le fichier journal externe, utiliser le menu '`Journal - Définir le fichier journal`'. Vous pourrez choisir un fichier avec un sélecteur ou entrer son nom. Les modifications seront ajoutées à ce fichier au cas où il existe déjà.

## 8.5 Définir le tag

La table '`_Log`' a un champ nommé '`_Label`' qui peut être librement défini par le projet. Le menu '`Journal - Définir le tag`' ouvre une fenêtre pour définir ce tag.

Il est aussi possible de fournir un tag à partir d'une fonction. La fonction doit être nommée `logLabel` et être appelée à chaque nouvelle entrée. Elle ne prend aucun argument et l'expression qu'elle retourne, convertie en texte, est utilisée comme tag. Pour d'autres informations à propos de cette fonction déclencheur, voir Section 16.29.6 [logLabel], page 159,

Notez que lorsque la fonction `logLabel` existe le menu '`Journal - Définir le tag`' est désactivé.

## 8.6 Importer un journal

Il est possible d'importer des entrées de journal depuis un fichier externe. C'est utile lorsque d'anciens entrées ont été supprimés du journal. Après avoir sélectionné le menu '`Journal -`

`Importer le journal`' une fenêtre s'ouvre que permet d'importer les enregistrements dans la table '`_Log`'. Voir Section 13.3 [Importing records], page 57, pour d'autres informations sur toutes les options d'importation.

## 8.7 Exporter le journal

Lorsque la table '`_Log`' grossit il est utile d'exporter les anciennes entrées dans un fichier externe. Pour ce faire, utilisez le menu '`Journal - Exporter le journal`'. Une fenêtre s'ouvre qui permet d'exporter les entrées de la table '`_Log`' vers un fichier. Voir Section 13.4 [Exporting records], page 57, pour d'autres informations sur toutes les options d'exportation.

## 8.8 Vider le journal

Pour nettoyer les entrées du journal, utilisez le menu '`Log - Vider le journal`'. Un cas typique pour nettoyer le journal est lorsque la table '`_Log`' devient trop grande. Soyez certain d'avoir exporté les entrées vers un fichier externe avant le vidage.

## 8.9 Appliquer les modifications

Le menu '`Journal - Appliquer les modifications`' permet d'importer les modifications depuis un fichier externe et de les appliquer au projet. Voir Section 13.3 [Importing records], page 57, pour une description de toutes les options d'import. Les modifications déjà présentes dans la table '`_Log`' du projet sont ignorées.

Cette fonction permet la restauration d'un projet éventuellement corrompu à partir d'une copie de sauvegarde en utilisant les modifications du fichier journal. L'idée est d'appliquer les modifications intervenues après la dernière sauvegarde, sur la base sauvegardée, en les important depuis le journal externe.

Il existe deux stratégies possibles qui peuvent être utilisées en fonction du mode de journalisation (voir Section 8.3 [Logging mode], page 42) :

- '`Dans un fichier`': lors de la création manuelle d'une copie de sauvegarde d'un projet, le fichier journal externe doit être effacé ou supprimé, de sorte que seules les nouvelles modifications apportées au projet seront présentes dans le fichier journal. Lors de l'application du fichier journal à la sauvegarde, toutes les modifications seront appliquées ;
- '`Dans les deux`': le fichier journal peut croître indéfiniment et n'a pas besoin d'être effacé lors de la création d'une copie de sauvegarde d'un projet. Les changements dans le fichier journal avant la copie de sauvegarde seront également dans la table '`_Log`' de la sauvegarde, et par conséquent, seuls les changements postérieurs à la sauvegarde seront appliqués.

## 8.10 Afficher le journal

Pour afficher le journal du projet, utiliser le menu '`Journal - Afficher le journal`'.

Si le mode de journalisation (voir Section 8.3 [Logging mode], page 42) est '`Dans le projet`' ou '`Dans les deux`' alors l'éditeur de requêtes s'ouvre avec une requête prédéfinie qui répertorie toutes les entrées du journal. Vous pouvez modifier la requête et son titre, et BeeBase se souviendra de la requête. Pour d'autres informations, voir Section 14.2 [Query editor], page 59.

Si le mode de journalisation est '`Dans un fichier`' alors le fichier journal externe est affiché (voir Section 7.1.3 [External viewer], page 35).

# 9 Édition d'enregistrement

Ce chapitre décrit comment ajouter, modifier, effacer et parcourir les enregistrements d'une table.

## 9.1 Objet actif

BeeBase utilise un curseur pour indiquer l'objet actif. Si l'objet actif est un objet texte, le curseur texte habituel apparaît, les autres objets sont encadrés par un cadre spécial. Il est possible de cycler sur les objets actifs avec les touches *Tab* ou *Maj-Tab*. En pressant les touches *Aide* ou *F1*, une visionneuse externe apparaît avec des informations utiles à propos de l'objet actif.

La table à laquelle l'objet actif appartient est appelée la *table active*. L'onglet d'une table peut devenir l'objet actif, cela assure qu'il est toujours possible de faire passer une table, en table active, même si celle-ci ne contient aucun objet activable.

Sous Windows, Mac OS et Linux chaque table dispose d'un menu contextuel avec les opérations permettant sa manipulation. Ce menu contextuel peut être ouvert en pressant le bouton droit de la souris n'importe où dans la fiche de la table (mais en dehors de tout autre élément graphique qui pourrait avoir son propre menu contextuel).

Sur Mac OS et Amiga les éléments de menu liés à la table font partie du menu global qui se trouve en haut de l'écran.

## 9.2 Ajout d'enregistrement

En sélectionnant le menu '`Table - Nouvel enregistrement`', un nouvel enregistrement est alloué dans la table active. Il est initialisé avec les valeurs par défaut pour chacun des champs. Il est également possible de dupliquer l'enregistrement courant de la table active en sélectionnant le menu '`Table - Dupliquer l'enregistrement`'.

Si un déclencheur pour la création d'enregistrement a été installé (voir Section 15.1.1 [Creating tables], page 65) alors celui-ci est appelé pour créer l'enregistrement. Pour plus de détails sur ce mécanisme, Voir Section 16.29.8 [New trigger], page 160.

## 9.3 Modification d'enregistrement

Pour modifier l'enregistrement courant d'une table, il suffit d'activer n'importe quel objet champ dans la fiche de la table et de saisir une nouvelle valeur. Pour les champs de type texte, entier, réel, date, heure et mémo il est possible d'utiliser les commandes d'édition habituelles.

Un objet champ peut être verrouillé en lecture seule. Dans ce cas il n'est pas possible de changer sa valeur (exception : les champs texte avec un bouton pop-up).

### 9.3.1 Champ texte avec bouton pop-up

Si un champ texte dispose d'un bouton attaché alors il est possible de cliquer sur ce bouton pour obtenir une requête pour saisir le contenu de la chaîne, Par ex. un sélecteur de fichier ou une liste de chaînes à sélectionner. Le bouton pop-up peut toujours être utilisé pour positionner la valeur du champ texte même si celui-ci est en lecture seule.

À droite du champ texte un autre petit bouton peut être présent. Une pression sur ce bouton lance la visionneuse externe pour afficher le fichier spécifié dans le champ texte.

### 9.3.2 Saisie de valeurs entières

Lors de la saisie d'un nombre entier, il est possible d'utiliser une notation octale (préfixe 0) ou hexadécimale (préfixe 0x) en plus de la notation décimale classique.

### 9.3.3 Saisie de valeurs booléennes

L'état d'activation d'un champ booléen peut être inversé en cliquant avec le bouton gauche de la souris ou en pressant la barre d'espace s'il est actif.

### 9.3.4 Saisie de valeurs de choix

Pour les champs de type choix, il est possible de sélectionner une valeur en cliquant sur le champ ou en utilisant les touches du curseur *`Haut`* et *`Bas`* pour parcourir toutes les entrées.

### 9.3.5 Saisie de valeurs de date

Les dates peuvent être saisies dans l'un des formats '`JJ.MM.AAAA`', '`MM/JJ/AAAA`' ou '`AAAA-MM-JJ`', où '`JJ`', '`MM`' et '`AAAA`' sont des valeurs de deux ou quatre chiffres représentant respectivement le jour, le mois et l'année de la date. Il est possible d'omettre la valeur de l'année d'une date, dans ce cas, l'année courante est utilisée.

En insérant une unique valeur entière, il est possible de spécifier une date relative à la date courante, Par ex. en entrant '`0`' la date du jour est utilisée, de même en entrant '`-1`' la date de la veille.

Un champ de date peut avoir un bouton pop-up à sa droite qui, lorsqu'il est pressé, ouvre un calendrier pour choisir une date.

### 9.3.6 Saisie de valeurs horaires

Le format de saisie de valeurs horaires est spécifié dans l'éditeur de structure (voir Section 15.3.3 [Field object editor], page 73). Les formats possibles sont '`HH:MM:SS`', '`MM:SS`' et '`HH:MM`' où '`HH`' représente les heures, '`MM`' les minutes, et '`SS`' les secondes.

Il est possible d'omettre certaines parties du format, Par ex. saisir '`6:30`' pour un format '`HH:MM:SS`' est interprété comme '`00:06:30`'. Lors de la saisie d'un nombre unique, il est considéré respectivement comme le nombre de secondes pour les formats '`HH:MM:SS`' et '`MM:SS`' et comme le nombre de minutes pour le format '`HH:MM`'.

### 9.3.7 Menu contextuel des mémos

Les champs Mémo possèdent un menu contextuel offrant des possibilités d'édition supplémentaires :

- '`Couper`', '`Copier`', et '`Coller`' permettent d'échanger des données avec le presse-papiers.
- '`Supprimer`' efface le texte sélectionné, tandis que '`Tout sélectionner`' permet de sélectionner tout le texte (Windows, Mac OS et Linux).
- '`Effacer`' efface tout le texte du mémo (Amiga).
- '`Annuler`' et '`Rétablir`' permettent d'annuler et rétablir les modifications du mémo (uniquement Amiga).

- '`Méthodes de saisie`' et '`Insérer un caractère de contrôle Unicode`' sont des éléments de menus spécifiques à GTK (Windows, Mac OS et Linux), se référer à la documentation GTK.
- Avec '`Ouvrir texte`' et '`Enregistrer le texte`' il est possible de charger et sauver le contenu du mémo depuis ou vers un fichier.
- '`Éditeur externe`' lance un éditeur externe pour modifier le mémo. Pour plus d'informations sur l'éditeur externe Voir Section 7.1.2 [External editor], page 34.

### 9.3.8 Menu contextuel de liste select-from-where

Les champs virtuels de type '`liste`' ont un menu contextuel contenant les entrées suivantes :

- '`Exporter au format texte`' pour exporter la liste dans un fichier texte (voir Section 14.3 [Exporting queries as text], page 60).
- '`Exporter au format PDF`' (sur la plupart des systèmes) pour exporter la liste dans un fichier PDF (voir Section 14.4 [Exporting queries as PDF], page 61).
- '`Imprimer`' pour imprimer la liste (voir Section 14.5 [Printing queries], page 61).

### 9.3.9 Saisie de valeurs de type Référence

Pour les champs de type Référence l'enregistrement référencé peut être entré avec un déroulant :

- À droite d'un champ Référence se trouve un bouton déroulant qui, s'il est pressé, ouvre une liste d'enregistrements. Choisir un enregistrement dans cette liste pour faire référence à cet enregistrement, le bouton '`Enregistrement initial`' permet de faire référence à la valeur NIL, et '`Enregistrement courant`' fait référence à l'enregistrement courant de la table référencée.
- Il est également possible de rechercher une entrée dans la table référencée en utilisant le clavier. Après avoir pressé la première touche un champ de saisie est ouvert permettant de saisir plus de caractères pour une recherche par motif. Après chaque touche pressée une recherche (sensible à la casse) est lancée et le premier enregistrement correspondant est sélectionné. La méthode de recherche peut être choisie dans l'objet affiché du champ (voir Section 15.3.3 [Field object editor], page 73) sous la même catégorie '`Recherche rapide`'. Vous pouvez utiliser le caractère '`*`' pour signifier une suite arbitraire de caractères et '`?`' pour exactement un caractère quelconque. L'utilisation des touches ***Haut*** et ***Bas*** permet de sélectionner respectivement la correspondance suivante ou précédente. Une entrée sélectionnée est mémorisée lors de son acceptation par la touche ***Entrée***. Si vous sortez de la fenêtre de recherche par tout autre moyen, p. ex. en pressant la touche ***Échap***, le champ reste inchangé et affiche son ancienne valeur.

Un champ Référence peut également être sélectionné par glisser-déposer d'une ligne depuis une liste d'un champ Virtuel vers un champ Référence. Si la ligne déplacée était générée depuis un enregistrement de la table référencée alors cet enregistrement est utilisé comme nouvelle référence d'enregistrement.

### 9.3.10 Saisie de valeurs NIL

Pour saisir la valeur NIL, il suffit de saisir une valeur invalide pour le type du champ, p. ex. entrer '`xyz`' dans un champ Entier, la valeur de ce champ est alors positionnée à NIL.

À noter cependant que tous les types de champ ne supportent pas la valeur NIL. Voir Section 5.6 [Table of field types], page 24, pour un aperçu des types de champ.

## 9.4 Suppression d'enregistrement

Le menu '`Table - Effacer l'enregistrement`' permet de supprimer l'enregistrement courant. Avant de supprimer l'enregistrement une requête de sécurité peut demander la confirmation. Il est possible d'activer et de désactiver cette requête dans les préférences (voir Section 7.2.2 [Confirm delete record], page 37).

Si un déclencheur a été installé pour la suppression d'enregistrement (voir Section 15.1.1 [Creating tables], page 65) alors celui-ci est appelé pour supprimer l'enregistrement. Pour plus d'informations sur ce mécanisme voir Section 16.29.9 [Delete trigger], page 160.

Il est également possible de supprimer tous les enregistrements d'une table en sélectionnant le menu '`Table - Supprimer tous les enregistrements`'. Seuls les enregistrements correspondants au filtre d'enregistrement de la table considérée sont supprimés. Avant la suppression, une confirmation peut apparaître si elle a été activée. Aucun déclencheur n'est appelé lors de la suppression de tous les enregistrements.

## 9.5 Parcourir les enregistrements

Pour afficher d'autres enregistrements que celui en cours, sélectionner l'un des sous-menus du menu '`Table - Atteindre l'enregistrement`'. Il est possible d'aller à l'enregistrement précédent, suivant, au premier, au dernier, sauter plusieurs enregistrements en arrière ou en avant ou même saisir le numéro d'enregistrement désiré. Dans ce contexte, le numéro d'enregistrement est le numéro affiché dans l'onglet correspondant (voir Section 5.8.3 [Panels], page 28). L'onglet peut également inclure deux boutons fléchés pour naviguer vers l'enregistrement précédent et suivant.

Le parcours d'enregistrements peut facilement être réalisé en utilisant les touches du curseur *`Haut`* et *`Bas`* en combinaison avec les touches *`Maj`*, *`Alt`*, et *`Ctrl`*. Toutes les combinaisons possibles sont listées dans les éléments du menu '`Table - Atteindre l'enregistrement`', et les combinaisons sont :

| | *`Alt`* | *`Maj+Ctrl`* | *`Maj+Alt`* |
|---|---|---|---|
| *`Haut`* | `Enregistrement précédent` | `Premier enregistrement` | `En arrière` |
| *`Bas`* | `Enregistrement suivant` | `Dernier enregistrement` | `En avant` |

## 9.6 Voir tous les enregistrements

Il est possible d'afficher tous les enregistrements d'une table avec le menu '`Table - Voir tous les enregistrements`' Ceci ouvre l'éditeur de requête avec une requête prédéfinie qui liste tous les enregistrements. de la table courante. Vous pouvez modifier la requête et son titre et BeeBase s'en rappellera. Pour plus d'informations, voir Section 14.2 [Query editor], page 59.

# 10 Filtre

Des filtres peuvent être utilisés pour masquer des enregistrements. Ce chapitre décrit les différents types de filtre disponibles et la façon de les utiliser.

BeeBase connaît deux types de filtres : les filtres d'enregistrement et les filtres de référence.

## 10.1 Filtre d'enregistrement

Un filtre d'enregistrement peut être positionné sur une table pour filtrer les enregistrements qui ne vous intéressent pas. Les enregistrements filtrés sont exclus de l'affichage de la fiche de table et par conséquent l'utilisateur ne peut ni les voir, ni les parcourir.

### 10.1.1 Expression de filtrage

Un filtre est défini en donnant une expression booléenne qui peut contenir des appels à des fonctions de programmation BeeBase. Pour chaque enregistrement de la table sur laquelle est spécifié le filtre, cette expression est évaluée. Si elle retourne NIL alors l'enregistrement est filtré, sinon l'enregistrement est inclus dans la fiche de table.

Chaque table peut avoir sa propre expression de filtrage.

### 10.1.2 Modifier les filtres

Pour modifier le filtre de la table courante, sélectionnez le menu '`Table - Changer le filtre`'. Cela ouvrira une fenêtre contenant :

- dans le titre de la fenêtre, le nom de la table sur laquelle vous appliquez le filtre.
- une liste de tous les champs de la table pouvant être utilisés dans l'expression de filtrage. Cette liste est placée sur la gauche de la fenêtre. Si vous double-cliquez sur un nom alors celui-ci sera inséré dans l'expression de filtrage à la position courante du curseur.
- une banque de boutons reprenant les fonctions de programmation et les opérateurs de BeeBase est placée sur la droite de la fenêtre. Cliquez sur l'un des boutons pour entrer la fonction correspondante dans l'expression de filtrage. Veuillez noter que la liste présentée des fonctions et opérateurs n'est pas exhaustive ; les autres fonctions de BeeBase qui ne sont pas présentes dans l'un des boutons doivent être saisies manuellement. Seules les fonctions BeeBase qui n'ont pas d'effet de bord peuvent être utilisées, p. ex. il n'est pas possible d'écrire des données dans un fichier au sein d'une expression de filtrage.
- un champ de saisie de chaînes de caractères pour entrer l'expression de filtrage. Les noms des champs, des fonctions et des opérateurs sont insérés ici. Vous pouvez également directement saisir votre expression de filtrage.
- deux boutons '`Ok`' et '`Annuler`' pour refermer la fenêtre.

Après avoir spécifié l'expression de filtrage, cliquez sur le bouton '`Ok`' pour quitter la fenêtre. L'expression saisie est alors compilée. En cas de compilation réussie, l'expression est appliquée à tous les enregistrements. Ceux pour qui l'expression booléenne n'est pas vérifiée sont retirés de l'affichage de la table.

Si l'expression ne compile pas, vous obtiendrez un message d'erreur dans la barre de titre de la fenêtre.

Un filtre peut être activé ou désactivé en cliquant sur le bouton '`F`' dans l'onglet de la table (si celui-ci est activé). Après la spécification d'une expression de filtrage sur une table, le filtrage de cette table est automatiquement activé.

Si vous (ré)activez un filtre alors tous les enregistrements de la table sont analysés pour vérifier s'ils répondent ou non au filtre.

Lorsqu'un filtre est activé et que vous changez la valeur d'un champ (employé dans le filtre) dans un enregistrement de cette table, la correspondance avec le filtre de cet enregistrement n'est pas réévaluée et reste inchangée.

Si vous allouez un nouvel enregistrement dans une table ayant un filtre actif, aucune vérification n'est faite sur le nouvel enregistrement quant à sa correspondance avec le filtre, il a son état de vérification automatiquement positionné à TRUE.

### 10.1.3 Exemples de filtre

Voici quelques exemples d'expression de filtrage valides :

- '`NIL`' filtre tous les enregistrements.
- '`TRUE`' ne filtre aucun enregistrement.
- '`0`' a le même effet que '`TRUE`' car pour BeeBase toute expression différente de NIL est équivalente à TRUE.
- '`(> Montant 100.0)`' n'affiche que les enregistrements dont le champ '`Montant`' est supérieur à 100,0 (on suppose ici que la table dispose d'un champ '`Montant`' de type décimal).
- '`(NOT (LIKE Nom "*x*"))`' filtre tous les enregistrements ayant la lettre '`x`' dans le champ '`Nom`' (de type chaîne).

Notez que le langage de programmation de BeeBase utilise une syntaxe similaire à celle de Lisp. Pour plus d'informations concernant le langage de programmation, voir Chapitre 16 [Programming BeeBase], page 81.

## 10.2 Filtre de référence

Les champs de type référence peuvent également avoir un comportement de filtre. Cela est utile par exemple pour construire une hiérarchie de tables. Comme exemple voyez le projet '`Albums`'.

Si le filtre d'un champ Référence est activé alors les fonctionnalités suivantes le sont aussi :

1. L'utilisateur ne peut accéder qu'aux enregistrements de la table pour lesquels la référence pointe sur l'enregistrement courant de la table référencée.
2. Si la table référencée change d'enregistrement courant, un nouvel enregistrement courant est également recherché et activé dans la table du champ Référence.
3. Lors de l'allocation d'un nouvel enregistrement, la référence est automatiquement positionnée à l'enregistrement courant de la table référencée.

Note: La suppression en cascade doit être implémentée manuellement (en utilisant un déclencheur de suppression).

N'utilisez pas les filtres de référence sur des graphes cycliques, p. ex. une table s'auto-référençant, car cela n'a pas beaucoup de sens.

# 11 Tri

Pour chaque table de votre base de données vous pouvez indiquer dans quel ordre les enregistrements devraient être affichés. Ce chapitre décrit comment spécifier un tri et quelles conséquences il a.

## 11.1 Tri vide

Par défaut, chaque table nouvellement créée à un tri vide. Ce qui signifie que si vous saisissez un nouvel enregistrement, il est inséré à la position courante (c'est-à-dire après l'enregistrement sélectionné dans la table). Si vous changez les champs d'un enregistrement existant, la position de l'enregistrement dans la table reste inchangée.

## 11.2 Tri par champs

C'est parfois utile de trier les enregistrements avec certains champs, comme par exemple avec un champ '`Nom`' si la table en a un.

Dans BeeBase, pour chaque table, vous pouvez indiquer une liste de champs par lesquels ses enregistrements devraient être triés. Tous les enregistrements sont d'abord triés avec le premier champ de cette liste. Si deux enregistrements sont équivalents pour un champ, c'est le champ suivant dans la liste qui déterminera l'ordre. Pour chaque champ vous pouvez indiquer si l'ordre des enregistrements triés est croissant ou décroissant.

Pour déterminer l'ordre des champs, les règles de la table suivante sont utilisées :

| **Type** | **Relation d'ordre** |
|---|---|
| `Integer` | `NIL < MIN_INT < ... < -1 < 0 < 1 < ... < MAX_INT` |
| `Choice` | `(ces valeurs sont traitées comme des nombres entiers)` |
| `Real` | `NIL < -HUGE_VAL < ... < -1.0 < 0.0 < 1.0 < ... < HUGE_VAL` |
| `String` | `NIL < "" < ... < "a" < "AA" < "b" < ...` |
| `Memo` | `(la comparaison d'une chaîne est insensible à la casse)` |
| `Date` | `NIL < 1.1.0000 < ... < 31.12.9999` |
| `Time` | `NIL < 00:00:00 < ... < 596523:14:07` |
| `Bool` | `NIL < TRUE` |
| `Reference` | `NIL < any_record` |
| | `(les enregistrements ne sont pas comparables entre eux pour le tri)` |

Si vous avez spécifié un tri pour une table, les enregistrements seront triés automatiquement à chaque fois que vous ajouterez un nouvel enregistrement ou que vous changerez un champ utilisé dans le tri d'un enregistrement.

## 11.3 Tri par fonction

Il est parfois utile d'avoir une commande de tri plus complexe qu'une simple liste de champs comme la précédente. Par exemple, une liste de champs ne peut pas contenir des champs de type référence. Il n'est donc pas possible de trier vos enregistrements selon un champ Référence pour la bonne et simple raison qu'avec les champs de type référence, BeeBase est incapable de maintenir à tout moment le tri de vos enregistrements (pour plus de détails, voir Section 16.29.10 [Comparison function], page 161, dans le chapitre de programmation de BeeBase).

Cependant, BeeBase offre également la possibilité d'indiquer une fonction de comparaison pour trier vos enregistrements. Vous pouvez indiquer n'importe quelle fonction que vous avez écrite en utilisant l'éditeur de programme de BeeBase. La fonction sera appelée et se positionnera sur les deux enregistrements avant de retourner la valeur que devrait refléter le tri des deux enregistrements. La fonction peut employer toutes les opérations pour comparer les enregistrements, ainsi elle peut par exemple comparer des enregistrements en utilisant des champs de référence. Pour plus d'information sur ce mécanisme, voir Section 16.29.10 [Comparison function], page 161.

Si vous décidez d'employer une fonction de comparaison pour trier une table, veuillez noter que BeeBase ne peut pas toujours détecter lorsque les enregistrements de la table doivent être triés à nouveau puisque les dépendances sont inconnues. Utilisez le menu '`Table - Réorganiser tous les enregistrements`' lorsque les enregistrements ne sont pas triés.

## 11.4 Changer l'ordre

Pour spécifier un tri, sélectionnez le champ de la table courante '`Table - Changer le tri`'. Une fenêtre s'ouvre contenant :

- Le nom de la table, dans le titre de la fenêtre.
- Une champ '`Type`' de tri où vous indiquez si la table doit être triée en utilisant une '`liste des champs`' ou en utilisant une '`fonction de comparaison`'. Selon le '`Type`' choisi, vous pouvez saisir des données dans les champs suivants.

Pour un tri utilisant une liste de champs, les champs suivants sont disponibles :

- Une liste de tous les champs de la table pouvant être employés dans la liste de tri. Cette liste est placée dans la partie gauche de la fenêtre. Si vous double-cliquez sur un nom alors ce nom sera inséré dans la liste de tri (à la position courante du curseur).
- La liste courante des champs utilisés pour le tri. Cette liste est placée dans la partie droite de la fenêtre. L'article situé en haut de cette liste est le premier champ de la liste de tri. Vous pouvez réarranger les articles en les glissant sur d'autres positions de la liste. Pour ajouter d'autres champs, glissez-les de la liste des champs à la liste de tri. Vous pouvez enlever un champ de la liste de tri en glissant son nom hors de la liste de tri et en le déposant sur la liste des champs.

  Chaque entrée dans la liste de tri a un symbole à sa gauche avec une flèche se dirigeant vers le haut ou vers le bas. Vous pouvez changer l'état de la flèche en la double-cliquant. Si une flèche se dirige vers le haut le tri pour ce champ sera ascendant, si elle se dirige vers le bas, il sera descendant.

Pour un tri utilisant une liste de champs, les champs suivants sont disponibles :

- Un champ '`Fonction de comparaison d'enregistrements`' où vous pouvez entrer le nom de la fonction qui devraient être appelée pour comparer deux enregistrements de la table. Vous pouvez utiliser le bouton menu à la droite du champ '`Fonction de comparaison d'enregistrements`' pour choisir un nom dans la liste de toutes les fonctions. Si vous laissez le champ vide, il n'y aura pas de tri. Pour plus d'information sur la façon d'employer une fonction de comparaison, y compris les arguments qui lui sont passés, voir Section 16.29.10 [Comparison function], page 161.

En outre, les boutons suivants existent :

- Un bouton '`Effacer`' qui désactive le tri (en oubliant la liste des champs).
- Deux boutons '`Ok`' et '`Annuler`' pour quitter la fenêtre.

Pour entrer un nouveau champ dans la liste de tri, choisissez le '`Type`' '`Liste des champs`' et appuyez sur le bouton '`Effacer`'. Utilisez alors le glisser/déposer comme décrit ci-dessus pour définir une nouvelle liste de champs. Si vous voulez pas de tri, il suffit juste de ne pas ajouter de champs dans la liste.

Quand vous avez spécifié l'ordre de tri, appuyez sur le bouton '`Ok`'. BeeBase réorganisera alors tous les enregistrements de la table.

## 11.5 Réordonner tous les enregistrements

Si vous pensez que certains enregistrements d'une table ne sont plus triés, par exemple en utilisant une fonction de comparaison pour le tri, alors vous pouvez utiliser le menu '`Table - Réordonner tous les enregistrements`' pour trier tous les enregistrements.

# 12 Recherche

Pour parcourir les enregistrements, vous pouvez utiliser la boite de recherche pour chercher un enregistrement particulier. La fonction de recherche utilise un motif de recherche (que vous devez fournir) et examine tous les enregistrements avec ce motif. Lorsqu'elle en trouve, l'enregistrement est affiché dans la fiche de table.

## 12.1 Boîte de recherche

Le menu '`Table - Rechercher`', ouvre la boîte de recherche, qui contient les éléments suivants :

- un champ texte pour saisir le motif de recherche. Les caractères '`*`' et '`?`' peuvent être utilisés comme jokers. Le caractère '`*`' remplace un nombre quelconque de n'importe quel caractère (y compris pas de caractère du tout), tandis que le caractère '`?`' remplace n'importe quel caractère mais exactement une fois.
- une option '`Sensible à la casse`' qui lorsqu'elle est cochée active la recherche de chaînes en différenciant majuscules et minuscules, sinon la recherche s'effectue sans distinction.
- une option '`Tous les champs`', qui lorsqu'elle est cochée, active la recherche de correspondances avec le motif spécifié dans tous les champs des enregistrements. Dans le cas contraire seul le champ actif au moment de l'ouverture de la boite de recherche est testé. Dans le cas où l'objet actif au moment de l'ouverture de la boite de recherche n'était pas un champ, celui-ci est vérifié et désactivé automatiquement.
- deux boutons radios pour le sens de la recherche, '`En avant`' et '`En Arrière`'.
- deux boutons radios pour déterminer à partir de quel enregistrement la recherche doit débuter, '`Premier/dernier enregistrement`' pour débuter la recherche au premier ou dernier enregistrement en fonction de la direction de recherche, '`Enregistrement courant`' pour débuter la recherche à partir de l'enregistrement courant.
- deux boutons '`Rechercher`' et '`Annuler`' pour sortir de la fenêtre.

Une fois le motif de recherche saisi et la boite de recherche fermée via le bouton '`Rechercher`', BeeBase commence à rechercher un enregistrement correspondant. La comparaison d'un champ avec le motif de recherche est toujours réalisée en mode texte, les champs qui ne sont pas de type texte sont donc auparavant convertis en chaînes de caractères.

Si un enregistrement correspondant est trouvé, il est affiché en tant qu'enregistrement courant dans la fiche de table, dans le cas contraire, un message '`Motif non trouvé`' est affiché.

Lorsque la recherche porte sur un champ qui est utilisé comme premier champ de tri et que le motif ne commence pas par un joker ('`*`' or '`?`'), un algorithme de recherche amélioré (recherche dichotomique) est employé qui prend en compte l'ordre des enregistrements, ce qui augmente significativement la vitesse de recherche.

## 12.2 Rechercher en avant / en arrière

Deux autres menus permettent de rechercher l'occurrence suivante et précédente du motif de recherche. Sélectionnez le menu '`Table - Rechercher suivant`' pour naviguer vers

le prochain enregistrement correspondant au motif de recherche, et '`Table - Rechercher précédent`' pour aller à l'enregistrement correspondant précédent.

## 12.3 Exemples de motif de recherche

voici quelques exemples de motif de recherche :

- '`Lassie`' recherche les enregistrements ayant la chaîne '`Lassie`' dans un des champs recherchés.
- '`*x*`' recherche les enregistrements ayant la lettre '`x`' dans un des champs recherchés.
- '`????`' recherche les enregistrements ayant exactement quatre caractères dans un des champs recherchés, p. ex. un enregistrement avec une entrée '`OVNI`'.

# 13 Importer et Exporter

Pour partager vos enregistrements avec d'autres systèmes de base de données, BeeBase offre un moyen d'importer et d'exporter les enregistrements depuis et vers d'autres bases de données. L'import et l'export sont réalisés par lecture et écriture de fichiers textes. Ces fichiers doivent respecter un format particulier décrit dans la section suivante.

## 13.1 Format de fichier

Pour importer des enregistrements dans BeeBase, tous les enregistrements d'une table doivent être disponibles dans un seul fichier texte. Si vous désirez importer des enregistrements de plusieurs tables, vous devez avoir plusieurs fichiers d'import (un par table).

Un fichier d'import est constitué de lignes et de colonnes. Les lignes sont séparées par un séparateur d'enregistrement, les colonnes par un séparateur de champ. Les délimiteurs peuvent être spécifiés dans les fenêtres d'import et d'export. Comme les champs des enregistrements eux-mêmes peuvent déjà contenir ces séparateurs, il est possible d'utiliser des guillemets anglais doubles autour des champs pour les protéger.

Un fichier d'import doit posséder la structure suivante :

- La première ligne contient le nom des champs, et pour chacun, un champ avec exactement le même nom doit exister pour la table dans laquelle sont importés les enregistrements. S'il existe un nom pour lequel il n'y a pas de champ correspondant dans la table, alors l'import échoue avec un message d'erreur.
- Les lignes suivantes contiennent chacune un enregistrement. Comme tous les champs doivent être spécifiés au format texte, ils sont automatiquement convertis dans le type du champ de destination. Pour les champs de type booléen, la valeur doit être soit NIL ou TRUE (peu importe la casse), autrement un message d'erreur est généré. Pour les champs de type choix, le libellé (label) exact doit être spécifié (la casse est importante ici). Pour les champs de type référence, il faut indiquer le numéro de l'enregistrement référencé (à partir de 1). Pour tous les autres types, la valeur NIL est utilisée si le champ ne peut être converti vers le type requis.
- Si vous utilisez des guillemets anglais doubles, tous les champs doivent être encadrés, y compris les noms des champs.

## 13.2 Exemple de fichier d'import

L'exemple suivant importe un fichier en utilisant respectivement `\n` et `\t` comme délimiteur d'enregistrement et délimiteur de champ, et des guillemets doubles autour de tous les champs. Le fichier peut être importé dans une table ayant les champs suivants :

- Nom (texte)
- NbEnfants (entier)
- Femme (booléen)
- Profession (choix)
- Notes (mémo)

```
"Nom" "NbEnfants" "Femme" "Profession" "Notes"
```

```
"Janet Jackson" "???" "TRUE" "Musicienne" "Dernier CD : The velvet rope"
"Bernt Schiele" "???" "NIL" "Scientifique" "Centres de recherches :
Robotique, Autonomie et vision par ordinateur"
"Gerhard" "0" "NIL" "Mécanique de précision" ""
```

## 13.3 Importer des enregistrements

Pour importer des enregistrements dans la table active, sélectionnez le menu '`Table - Importer des enregistrements`'. Une fenêtre s'ouvre contenant les éléments suivants :

- un champ texte pour saisir le nom du fichier d'import. À droite de ce champ il y a trois boutons : le premier permet de sélectionner un fichier, le second lance la visionneuse externe pour inspecter le fichier sélectionné, et le troisième lance un éditeur pour modifier le fichier.
- deux champs texte pour saisir respectivement les délimiteurs d'enregistrement et de champ. Vous pouvez saisir un caractère ou un code de contrôle tel que `\n`, `\t`, `\f`, `\???` (code octal) ou `\x??` (code héxa). Les délimiteurs doivent être des caractères ASCII 7 bits (de `\x01` à `\x7F` inclus).
- une case à cocher '`Guillemets anglais doubles`' pour indiquer que les champs sont protégés par des guillemets anglais doubles.
- deux boutons '`Importer`' et '`Annuler`' pour fermer la fenêtre.

Si vous appuyez sur le bouton '`Importer`', BeeBase charge le fichier spécifié et importe tous les enregistrements trouvés. Si tout se passe correctement après le processus d'import, BeeBase demande si vous voulez vraiment ajouter les enregistrements importés dans la table. À ce point il est encore possible d'annuler l'opération. Les enregistrements écrasés, cependant, ne peuvent être récupérés qu'en restaurant le projet.

Si une erreur survient lors de la lecture du fichier d'import, un message d'erreur est affiché.

Si vous avez besoin d'un mécanisme d'import plus sophistiqué, il est recommandé d'écrire sa propre routine d'import dans un programme BeeBase.

## 13.4 Exporter des enregistrements

Pour exporter les enregistrements de la table active, sélectionnez le menu '`Table - Exporter des enregistrements`'. Une fenêtre s'ouvre contenant les éléments suivants :

- un champ texte pour saisir le nom du fichier d'export avec à droite un bouton pour sélectionner un fichier.
- deux champs texte pour saisir respectivement les délimiteurs d'enregistrement et de champ. Vous pouvez saisir un caractère ou un code de contrôle tel que `\n`, `\t`, `\f`, `\???` (code octal) ou `\x??` (code héxa). Les délimiteurs doivent être des caractères ASCII 7 bits.
- une case à cocher '`Guillemets anglais doubles`' pour indiquer que les champs doivent être protégés par des guillemets anglais doubles.
- deux boutons '`Exporter`' et '`Annuler`' pour fermer la fenêtre.

Si vous appuyez sur le bouton '`Importer`', BeeBase ouvre le fichier spécifié et y écrit les enregistrements en y incluant une ligne d'entête contenant le nom des champs. La fonction d'export écrit toujours tous les champs de la table dans le fichier d'export.

Pour un mécanisme d'export personnalisé, vous pouvez utiliser l'éditeur de requêtes de BeeBase (voir Chapitre 14 [Data retrieval], page 59) ou écrire votre propre routine d'export dans un programme BeeBase.

# 14 Traitement des données

Deux méthodes sont utilisables dans BeeBase pour le traitement des données : par programmation ou depuis éditeur de requêtes.

La méthode par programmation vous permet de créer des boutons dans les masques de tables qui, sur pression, appelleront des fonctions programmées. L'utilisation de cette méthode est décrite dans le chapitre concernant l'éditeur de structure (voir Chapitre 15 [Structure editor], page 65) et dans le chapitre à propos de la programmation dans BeeBase (voir Chapitre 16 [Programming BeeBase], page 81).

Ce chapitre décrit l'utilisation de l'éditeur de requêtes, une fenêtre où vous pouvez entrer des requêtes et consulter le résultat dans une fenêtre défilante.

## 14.1 Requêtes Select-from-where

BeeBase utilise un système de requête du type select-from-where comme dans les bases de données SQL. La requête vous permet d'afficher le contenu des enregistrements d'une ou plusieurs tables. Seuls les enregistrements qui correspondent à un certain critère se retrouvent dans le résultat. La syntaxe (incomplète) d'une requête select-from-where est

```
SELECT exprliste FROM tableliste [WHERE test-expr]
[ORDER BY orderliste]
```

où *exprliste* est une liste d'expressions, séparées par des virgules, à afficher (souvent les noms de champs) ou une simple étoile `*` qui correspond à tous les champs des tables spécifiées, *tableliste* est une liste de tables, séparées par des virgules, dans lesquelles les enregistrements sont examinés, *test-expr* est l'expression qui est testée pour chaque groupe d'enregistrements qui doit être inclus dans le résultat, et *orderlist* est une liste de champs, séparés par des virgules, qui déterminent l'ordre pour afficher le résultat. Notez que les clauses WHERE et ORDER BY sont optionnelles ; ce qui est indiqué par les crochets `[]`.

Par exemple, la requête

```
SELECT * FROM table
```

affiche le contenu des champs de tous les enregistrements de la table spécifiée.

```
SELECT champ1 FROM table WHERE (LIKE champ2 "*Madonna*")
```

affiche les valeurs du champ *champ1* de tous les enregistrements de *table* pour laquelle le contenu du champ *champ2* contient le mot '`Madonna`'.

Pour plus d'informations à propos des requêtes select-from-where et notamment sa syntaxe complète, voir Chapitre 16 [Programming BeeBase], page 81, pour plus d'exemples voir Section 14.6 [Query examples], page 63.

## 14.2 Éditeur de requêtes

Pour saisir et exécuter des requêtes, ouvrez l'éditeur de requêtes en sélectionnant le menu '`Programme - Requêtes`'. L'éditeur de requêtes peut gérer plusieurs requêtes mais une seule est affichée à la fois. La fenêtre de l'éditeur de requêtes contient les éléments suivants :

- un champ de saisie de texte avec un bouton pop-up attaché. Le champ texte indique le nom de la requête en cours. En pressant le bouton, apparaissent une liste avec d'autres noms de requêtes ainsi que d'autres boutons. Vous pouvez sélectionner une des requêtes

affichées pour en faire la requête courante. Pressez le bouton '`Nouveau`' pour créer une nouvelle requête. Pressez le bouton '`Dupliquer`' pour faire une copie de la requête sélectionnée. Cliquez sur le bouton '`Trier`' pour trier la liste des requêtes, ou pressez le bouton '`Supprimer`' pour effacer la requête en cours. Pour quitter la fenêtre sans rien modifier, cliquez à nouveau sur le bouton des requêtes.

- une liste de choix qui permet d'assigner une requête sur une table. À l'assignation d'une table, BeeBase lance cette requête lorsque l'utilisateur sélectionne le menu '`Table - Afficher tous les enregistrements`' de la table correspondante.
- un bouton '`Exécuter`' qui compile et exécute le programme de requête et affiche le résultat dans la fenêtre déroulante.
- un bouton '`Export`' qui ouvre une fenêtre (voir Section 14.3 [Exporter des requêtes en texte], page 60) pour exporter le résultat de la requête dans un fichier texte.
- un bouton '`PDF`' (sur la plupart des systèmes) qui ouvre une fenêtre (voir Section 14.4 [Exporter des requêtes en PDF], page 61) pour exporter le résultat de la requête dans un fichier PDF.
- un bouton '`Imprimer`' qui ouvre une requête (voir Section 14.4 [Exporter des requêtes en PDF], page 61) pour imprimer le résultat.
- un bouton '`Imprimer`' qui ouvre une fenêtre (voir Section 14.5 [Printing queries], page 61) pour imprimer les résultats.
- un champ de saisie pour saisir la requête. Vous saisissez ici une requête de type select-from-where. Cependant, il est possible d'entrer une expression du langage de programmation de BeeBase. cela peut être utile si vous voulez faire des calculs simples ou modifier certains champs d'une table en utilisant un programme simple. Veuillez noter que BeeBase entoure automatiquement votre expression avec une paire de parenthèses. Vous pouvez donc omettre d'utiliser les plus extérieures.
- un affichage en liste qui montre le résultat de la requête en cours. L'affichage est formaté en lignes et colonnes. La ligne de titre contient les noms des champs de la requête select-from-where (le plus souvent les noms des champs). Sur Les lignes suivantes est affiché le résultat de la requête, un jeu d'enregistrement par ligne. Chaque champ est affichée dans sa propre colonne. En cliquant sur le titre d'une colonne vous pouvez trier la liste en fonction de cette colonne. Un second clic sur la même colonne inverse l'ordre de tri. Sur Amiga vous pouvez sélectionner une seconde colonne en maintenant la touche *`Maj`* enfoncée tout en cliquant sur le titre. Si vous double-cliquez un élément de la liste, et que cet élément était généré directement à partir d'un enregistrement, alors cet enregistrement est affiché dans le masque de table correspondant. C'est une façon simple d'atteindre un enregistrement dans son masque de table.

L'éditeur de requêtes est une fenêtre non modale. Cela signifie que vous pouvez laisser l'éditeur de requêtes ouvert alors que vous travaillez avec le reste de l'application. Vous pouvez à tout moment fermer l'éditeur de requêtes, en cliquant sur le bouton de fermeture sur la barre de titre.

## 14.3 Exporter des requêtes en texte

Vous pouvez exporter le résultat d'une requête select-from-where vers un fichier texte en appuyant sur le bouton '`Exporter`'. Une fenêtre s'ouvrira contenant :

- un champ texte pour indiquer le fichier d'export. À droite du champ un bouton ouvre la fenêtre de sélection de fichier.
- deux champs texte pour indiquer les délimiteurs d'enregistrement et de champ. Vous pouvez saisir un caractère unique ou une séquence d'échappement en tapant `\n`, `\t`, `\f`, `\???` (code octal), ou `\x??` (code hexa.).
- un champ '`Guillement anglais double`' qui peut être coché pour indiquer que les champs doivent être protégés par des guillemets anglais doubles.
- deux boutons '`Exporter`' et '`Annuler`' pour quitter la fenêtre.

Après avoir pressé le bouton '`Exporter`', BeeBase ouvrira le fichier indiqué et écrira le résultat de la requête en incluant une ligne d'entête. Les champs sont écrits dans l'ordre des colonnes de la liste.

## 14.4 Exporter des requêtes en PDF

Sur Windows, Mac OS X, Linux et MorphOS vous pouvez exporter les résultats d'une requête vers un fichier PDF en pressant le bouton '`PDF`'.

Une fenêtre d'impression contient les options ci-dessous :

- un champ texte pour indiquer le fichier d'export. À droite du champ un bouton ouvre la fenêtre de sélection de fichier.
- une liste de sélection du format de page.
- une option pour choisir l'orientation de la page : ('`Portrait`' ou '`Paysage`').
- un champ pour indiquer une police de caractères et une taille. Sur Windows, Mac OS X et Linux, vous pouvez cliquer sur le bouton pop-up button à droite pour sélectionner la police via une boîte de dialogue. Sur MorphOS vous sélectionnez la police directement dans la liste et entrez la taille dans le champ de droite. Il est possible d'entrer une fraction, p. ex. '`10,5`'.
- un champ texte pour décrire un texte facultatif en entête. L'entête est imprimé en haut de chaque page. Laisser vide si vous ne désirez pas d'entête.
- un champ texte pour décrire un pied de page facultatif. Le pied de page est imprimé au bas de chaque page avec le numéro de page.
- un aperçu pour connaître le nombre de pages et vérifier que le texte tient dans la page.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre d'impression.

Après avoir pressé le bouton '`Créer un PDF`', BeeBase ouvrira le fichier indiqué et écrira le résultat de la requête avec une ligne d'entête contenant la liste des champs. Les champs apparaissent dans l'ordre des colonnes de la liste.

## 14.5 Impression de requêtes

Après avoir exécuté une requête, vous pouvez imprimer le résultat en cliquant sur le bouton '`Imprimer`' dans l'éditeur de requêtes.

Sur Windows, Mac OS X et Linux ceci ouvre la fenêtre d'impression standard.

Si vous utilisez la version GTK de BeeBase, la fenêtre contient une page personnalisée de '`polices de caractères`' où vous pouvez choisir une police et activer la contraction du

texte pour ajuster le contenu à la largeur et l'orientation de page. Après avoir pressé le bouton '`Imprimer`' les résultats de la requête sont envoyés à l'imprimante sélectionnée.

Sur MorphOS le bouton '`Imprimer`' ouvre une fenêtre semblable d'export des requêtes en PDF (voir Section 14.4 [Exporting queries as PDF], page 61).

La fenêtre contient les options suivantes :

- une liste de sélection du format de page.
- une option pour choisir l'orientation de la page : ('`Portrait`' ou '`Paysage`').
- une option pour le choix de la police de caractères et un champ pour saisir la taille désirée. Il est possible d'entrer une fraction, p. ex. '`10,5`'.
- un champ texte pour décrire un texte facultatif en entête. L'entête est imprimé en haut de chaque page. Laisser vide si vous ne désirez pas d'entête.
- un champ texte pour décrire un pied de page facultatif. Le pied de page est imprimé au bas de chaque page avec le numéro de page.
- un aperçu pour connaître le nombre de pages et vérifier que le texte entre dans la page.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre d'impression.

Après avoir pressé le bouton '`Ok`', BeeBase générera un fichier temporaire qu'il ouvrira dans un lecteur externe (voir Section 7.1.3 [External viewer], page 35). Vous pouvez alors utiliser les fonctionnalités de votre lecteur pour envoyer le résultat de la requête à l'imprimante.

Sur les autres systèmes Amiga, la fenêtre d'impression contient les options ci-dessous :

- un champ '`Délimiteur`' où vous précisez comment les colonnes doivent être séparées. '`Espaces`' place des caractères espaces entre chaque champ. La justification est faite à gauche ou à droite en fonction du type du champ (les nombres sont justifiés à gauche, le texte à droite). '`Tabulations`' insère un caractère tabulation entre les colonnes. Cela peut être utile si vous voulez utiliser la fenêtre d'impression pour exporter des enregistrements (voir ci-dessous). '`Personnaliser`' vous permet de préciser un délimiteur personnalisé à afficher entre les champs.
- un champ '`Police de caractères`' où vous précisez la police de caractères d'impression. '`NLQ`' signifie « qualité proche d'une lettre » qui doit imprimer dans une meilleure qualité que '`Brouillon`'.
- un champ '`Taille`' où vous définissez la taille des caractères. '`Pica`' imprime avec une police de grande taille (10 cpi), '`Elite`' de taille moyenne (12 cpi) et '`Condensé`' de petite taille (17 cpi).
- un champ texte '`Séquence d'initialisation`' qui vous permet de préciser votre imprimante. Le contenu de ce champ est directement envoyé à l'imprimante à l'ouverture de celle-ci. Par exemple, vous pouvez utiliser '`\33c`' comme séquence pour réinitialiser votre imprimante.
- un champ '`Indentation`' où vous pouvez préciser le nombre d'espaces qui seront utilisés pour indenter chaque ligne en sortie.
- un champ '`Entête`' qui, s'il est coché, imprime les noms de champ sur la première ligne.
- un champ '`Codes d'échappement`'. S'il n'est pas coché l'impression de tous les codes d'échappement est supprimé, donc les paramétrages des champs '`Police de caractères`' et '`Taille`' sont ignorés et le contenu de la '`Séquence`

d'initialisation' n'est pas imprimée. Supprimer l'impression de tous les codes d'échappement est utile si vous voulez générer un fichier ASCII, par exemple pour exporter des enregistrements.

- un champ 'Guillemets' qui, s'il est coché, entoure chaque champ de guillemets anglais doubles.
- un champ 'Après impression' où vous choisissez comment doit se terminer l'extraction. 'Saut de page' imprime un caractère saut de page \f. 'Saut de ligne' imprime un certain nombre de sauts de ligne \n. Le nombre de sauts de ligne peut être saisi dans le champ texte à la droite du bouton 'Sauts de ligne'. 'Rien' n'imprime rien sur l'imprimante.
- un champ Texte 'Sortie' avec un bouton pop-up. Vous pouvez utiliser le bouton pour ouvrir une requête de fichier et choisir ou saisir un nom de fichier. Pour envoyer la sortie vers l'imprimante entrez |lpr (Linux) ou PRT: (Amiga). Pour d'autres noms de fichiers spéciaux, voir Section 7.2.13 [Program output file], page 39.
- deux boutons 'Ok' et 'Annuler' pour fermer la fenêtre d'impression.

Quand vous en avez terminé avec tous les paramètres, cliquez sur le bouton 'Ok' pour lancer le travail d'impression.

## 14.6 Exemples de requêtes

Voici plusieurs exemples de requêtes pour vous donner une idée de la puissance des requêtes select-from-where.

Supposez que nous ayons deux tables 'Personne' et 'Chien'. 'Personne' a les champs suivants : 'Nom', un champ Entier 'Age', et deux champs de référence 'Pere' et 'Mere' qui se réfèrent aux enregistrements père et mère de la table 'Personne'.

La table contient les enregistrements suivants :

```
      Nom       Age      Pere  Mere
      --------------------------------
p1:   Steffen    26     p2      p3
p2:   Dieter     58     NIL     NIL
p3:   Marlies    56     NIL     NIL
p4:   Henning    57     NIL     NIL
```

'Chien' a un champ Texte 'Nom', un champ de type liste 'Couleur' et un champ de référence 'Propriétaire' qui se réfère au propriétaire dans la table 'Personne'. La table contient les enregistrements suivants :

```
      Nom      Couleur   Propriétaire
      ----------------------
d1:   Boy       blanc   p3
d2:   Streuner  gris    NIL
```

Avec ces données, on peut exécuter les requêtes select-from-where suivantes :

SELECT * FROM Personne

donne :

```
Name    Age  Pere Mere
-------------------------
Steffen  26 Dieter Marlies
Dieter   58
Marlies  56
Henning  57
```

(Pour les champs de référence le champ ‘`Nom`’ de l’enregistrement référencé est affiché.)

```
SELECT Nom "Enfant", Age,
     Pere.Nom "Pere", Pere.Age "Age",
     Mere.Nom "Mere", Mere.Age "Age"
FROM Personne WHERE (AND Pere Mere)
```

donne :

```
Enfant   Age Pere  Age Mere  Age
---------------------------------
Steffen  26 Dieter  58 Marlies  56
```

```
SELECT Nom, Couleur,
     (IF Propriétaire Proprietaire.Nom "Pas de proprietaire") "Propriétaire"
FROM Chien
```

donne :

```
Nom    Couleur  Propriétaire
------------------------
Boy      blanc  Marlies
Streuner gris   Pas de propriétaire
```

```
SELECT a.Nom, a.Age, b.Nom, b.Age FROM Personne a, Personne b
WHERE (> a.Age b.Age)
```

donne :

```
a.Nom  a.Age  b.Nom  b.Age
--------------------------
Dieter     58 Steffen    26
Marlies    56 Steffen    26
Henning    57 Steffen    26
Dieter     58 Marlies    56
Henning    57 Marlies    56
Dieter     58 Henning    57
```

# 15 Éditeur de structure

BeeBase dispose de deux modes de fonctionnement différents : le mode édition d'enregistrement où il est possible de saisir et parcourir les enregistrements, et le mode édition de structure où se définit la structure de la base, c'est à dire les tables, les champs et l'apparence d'un projet. Ce chapitre décrit l'éditeur de structure et explique comment gérer la structure d'un projet.

Pour basculer du mode édition d'enregistrement à celui d'édition de structure, il faut sélectionner l'élément '`Éditeur de structure`' dans le menu '`Projet`', ce qui ferme toutes les fenêtres et ouvre la fenêtre de l'éditeur de structure. Pour revenir au mode édition d'enregistrement, il faut sélectionner le menu '`Projet - Fermer l'éditeur de structure`' ou simplement fermer l'éditeur de structure en cliquant sur le gadget de fermeture dans la barre de titre de la fenêtre.

La fenêtre de l'éditeur de structure est divisée en trois parties : la partie supérieure gauche composée du groupe '`Tables`' servant à gérer les tables du projet, la partie inférieure gauche quant à elle occupée par le groupe '`Champs`' permet de gérer les champs d'une table, et pour finir la partie droite est occupée par un groupe '`Affichage`' pour gérer les éléments graphiques du projet (interface).

## 15.1 Gestion des tables

Depuis le groupe '`Tables`' de l'éditeur de structure, vous pouvez créer, changer, supprimer et trier les tables.

### 15.1.1 Création de tables

Pour créer une nouvelle table, appuyez sur le bouton '`Nouveau`' du groupe '`Tables`' ce qui ouvrira la fenêtre '`Nouvelle table`' contenant :

- un champ de texte pour nommer la table. Chaque table doit avoir un nom unique commençant par une lettre majuscule suivie d'autres lettres, de chiffres ou de tirets bas. Les caractères non-ASCII (tréma, double point) ne sont pas autorisés. En revanche, il est tout à fait possible d'utiliser des caractères non-ASCII dans l'interface utilisateur de la table.
- un champ '`Nombre d'enregistrements`' dans lequel vous définissez combien d'enregistrements comportera la table. '`Illimité`' signifie que la table peut contenir un nombre infini d'enregistrements, '`Exactement un`' signifie que la table ne peut comporter qu'un seul enregistrement. Ce champ est utile pour contrôler le projet (voir Section 5.2 [Tables], page 20).
- un champ '`Déclencheurs`' dans lequel vous pouvez définir le nom de deux fonctions. Dans le champ '`Nouveau`' vous entrez le nom de la fonction à appeler à chaque fois que l'on veut créer un nouvel enregistrement, alors que le champ '`Suppression`' contient le nom de la fonction à appeler à chaque suppression d'enregistrement. Vous pouvez également utiliser les boutons déroulants situés à droite des champs de texte pour choisir à partir d'une liste de toutes les fonctions celle à appeler. Si vous laissez ces champs vides, alors ce sont les actions par défaut qui seront exécutées (les enregistrements sont créés automatiquement et les enregistrements sont effacés après une éventuelle requête de confirmation). Pour de plus amples informations concernant l'utilisation de

ces déclencheurs, ainsi que le passage d'arguments, voir Section 16.29.8 [New trigger], page 160, et Section 16.29.9 [Delete trigger], page 160.

- une case à cocher '`Compter les modifications`'. Si active, chaque création ou suppression d'un enregistrement est compté comme un changement du projet. Dans le cas contraire aucun changement de la table (ou d'un de ses champs) n'est compté.
- une case à cocher '`Tracer les modifications`'. Si active, chaque création ou suppression d'un enregistrement est enregistré dans le journal du projet. Dans le cas contraire aucun changement de la table (ou d'un de ses champs) n'est tracé.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Une fois que vous avez effectué tous les réglages, pressez le bouton '`Ok`' pour créer la nouvelle table. Si vous avez fait une erreur, comme entrer un nom invalide, un message s'affichera et vous donnera des informations sur l'erreur. À l'inverse si tout se passe correctement, la fenêtre '`Nouvelle table`' se fermera et la nouvelle table s'affichera dans la liste des tables de l'éditeur de structure.

### 15.1.2 Modification de tables

Après avoir créé une table, vous pouvez toujours la modifier. Il vous suffit de double cliquer sur le nom de la table et la fenêtre '`Modifier la table`' apparaîtra. Cette fenêtre est similaire à celle de création de table (voir Section 15.1.1 [Creating tables], page 65) et vous permet de modifier les champs en saisissant une nouvelle valeur.

Quand vous avez effectué toutes vos modifications appuyez sur le bouton '`Ok`' pour fermer la fenêtre.

Notez que vous ne pouvez pas changer le nombre d'enregistrements d''`Illimité`' en '`Exactement un`' si la table contient déjà plus d'un enregistrement.

### 15.1.3 Suppression de tables

Pour supprimer une table, sélectionnez son nom dans la liste de l'éditeur de structure, puis cliquez sur le bouton '`Supprimer`' sous la liste. Avant de supprimer définitivement la table, une requête de confirmation sera ouverte. Si vous confirmez en appuyant sur le bouton '`Supprimer`', la table sera irrémédiablement supprimée.

Un problème se pose si la table est utilisée ailleurs dans un projet. Dans ce cas, la table ne peut pas être supprimée directement. Toutes les références à cette table doivent être éliminées du programme. Si la table à supprimer est utilisée dans un projet, l'éditeur apparaîtra et affichera la première référence de la table. Vous devez alors modifier le programme de manière à éliminer toute référence à la table. Après avoir éliminé une référence, vous pouvez aller à la prochaine en cliquant sur le bouton '`Compiler`'. Vous pouvez à tout moment annuler l'opération en cliquant sur le bouton '`Rétablir`' et fermer l'éditeur.

### 15.1.4 Tri des tables

Vous pouvez trier les tables dans le champ '`Tables`' de l'éditeur de structure de plusieurs façons. Vous pouvez les arranger manuellement par glisser/déposer ou vous pouvez utiliser le bouton '`Tri`' sous la liste pour les classer par ordre alphabétique ('`Tri 1`') ou bien encore en fonction de l'ordre de la liste d'affichage ('`Tri 2`').

## 15.2 Gestion des champs

Avec le champ '`Champs`' de l'éditeur de structure, vous pouvez créer, copier, modifier, effacer et trier les champs de la table sélectionnée dans '`Tables`'.

### 15.2.1 Création de champs

Pour créer un nouveau champ dans la table active, cliquez sur le bouton '`Nouveau`' dans le champ '`Champs`'. Cette action ouvrira la fenêtre '`Nouveau champ`' qui contient les éléments suivants :

- un champ de texte pour entrer le nom du champ. Chaque champ d'une table doit avoir un nom unique commençant par une lettre majuscule suivie d'autres lettres, chiffres ou tiret bas. Les caractères non-ASCII (les trémas par exemple) ne sont pas autorisés. Cependant il est toujours possible d'utiliser pour les champs (dans l'interface utilisateur) n'importe quel caractère y compris non-ASCII.
- une liste '`Type`' dans laquelle vous précisez le type de champ. Pour plus de précision sur les types de champ, voir Section 5.5 [Field types], page 21.
- une section sous le champ '`Type`' pour préciser les réglages spécifiques au type de champ. Pour plus de précisions sur cette section, voir Section 15.2.2 [Type specific settings], page 67.
- un champ '`Déclencheur`' dans lequel vous donnez le nom d'une fonction qui sera appelée chaque fois que l'utilisateur voudra modifier le contenu du champ dans un enregistrement. Vous pouvez utiliser le bouton contextuel (ou bouton déroulant) à droite du champ de texte pour sélectionner un nom parmi une liste de noms de fonction. Si vous laissez ce champ vide, alors l'action par défaut sera exécutée, ce qui veut dire que la valeur entrée sera simplement stockée dans le champ. Pour plus de précision sur le déclenchement de fonction, y compris comment passer des arguments, voir Section 16.29.11 [Field trigger], page 162.
- une case à cocher '`Tenir compte des modifications`'. Si elle est cochée, toute modification dans un champ d'un enregistrement, sera considérée comme une modification du projet. Décochez cette case si vous voulez ignorer les modifications de champ.
- un champ '`Tracer les modifications`', qui, si coché, trace toute modification de champ d'un enregistrement . Notez que cet élément est actif à la condition que l'option '`Tracer les modifications`' d'un élément de table ait été également activé (voir Section 15.1.1 [Creating tables], page 65).
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Lorsque vous avez effectué vos réglages, appuyez sur le bouton '`Ok`' pour créer le nouveau champ. Si vous avez fait une erreur tel qu'un nom invalide, un message s'affichera et vous donnera des informations sur l'erreur. À l'inverse si tout se passe correctement, la fenêtre '`Nouveau champ`' se fermera et le nouveau champ s'affichera dans la liste des champs de l'éditeur de structure.

### 15.2.2 Réglages liés au type de champ

Voici les modifications spécifiques aux types de champ que l'on peut effectuer :

- Pour les champs de type Texte :

  - un champ de type Entier '`Taille maximum`' pour définir la longueur maximale pour ce champ.
  - un champ de type Texte '`Valeur par défaut`' pour définir la valeur par défaut de ce champ. On peut entrer n'importe quel texte dans la limite de la longueur maximale.

- Pour les champs de type Entier, Réel, Date et Heure :
  - un champ '`Valeur par défaut`' dans lequel vous définissez la valeur par défaut de ce champ. Vous avez le choix entre '`NIL`' et '`Autre`'. Si vous utilisez '`Autre`' vous devez spécifier une valeur initiale dans le champ de texte à droite.
  - un champ de type Texte '`chaîne NIL`' dans lequel vous définissez le texte qui sera affiché lorsque le champ contiendra la valeur NIL.
- Le réglage spécifique aux champs de type Booléen comprend un champ '`Valeur par défaut`' dans lequel vous entrez la valeur par défaut : soit '`NIL`' soit '`TRUE`'.
- Pour les champs de type Choix :
  - un bouton '`Édition des entrées`' pour ouvrir la fenêtre '`Édition des entrées`' et donnez un nom au champ de type Choix (voir Section 15.2.3 [Label editor], page 68).
  - un champ '`Valeur par défaut`' dans lequel vous entrez la valeur par défaut.
- Pour les champs de type Référence :
  - une liste de toutes les tables pour sélectionner celle vers laquelle pointera le champ. En cliquant sur la table, le champ ira vers la référence.
  - un champ '`Affichage auto`'. S'il est coché, la table de référence sera mise à jour automatiquement en fonction de l'enregistrement de référence, chaque fois que l'utilisateur passera à un autre enregistrement.
  - un champ '`Filtrer ?`'. S'il est coché, le filtre de référence de ce champ sera activé. Voir Section 10.2 [Reference filter], page 50, pour plus de précision.

  Les champs de type Référence ont toujours NIL comme valeur par défaut.
- Le réglage spécifique aux champs de type Virtuel concerne
  - un champ de texte '`Calculer`' dans lequel vous donnez le nom de la fonction à lancer pour calculer la valeur du champ. Vous pouvez vous servir du bouton déroulant pour sélectionner un nom parmi une liste de tous les noms de fonction. Veuillez voir Section 16.29.12 [Programming virtual fields], page 162, pour en savoir plus sur l'utilisation de cette fonction.
  - un champ de type Texte '`chaîne NIL`' dans lequel vous définissez le texte qui sera affiché lorsque le champ contiendra la valeur NIL.
- Les champs de type Mémo et Bouton n'ont pas de réglages propres. La valeur par défaut des champs de type Mémo est une ligne de texte vierge.

### 15.2.3 Éditeur d'entrée

L'éditeur d'entrées intervient à chaque fois que vous devez définir une liste comme par exemple une liste de noms pour un champ de type Choix. L'éditeur se compose d'une fenêtre qui contient :

- une liste montrant les entrées existantes. Cliquer sur une entrée la sélectionner. Elle sera visible dans le champ de texte sous la liste. Vous pouvez utiliser le glisser/déposer pour réarranger les entrées.
- un champ de texte '`Entrée`' qui affiche l'entrée sélectionnée et permet également de la modifier. Les modifications ne seront prises en compte que lorsque vous appuyez sur la touche *`Entrée`*. S'il n'y a aucune entrée sélectionnée, la pression sur *`Entrée`* ajoutera une nouvelle entrée à la liste.
- un bouton '`Nouveau`' qui désélectionne l'entrée courante, ce qui permet d'ajouter de nouvelles entrées dans le champ de texte '`Entrée`'.
- un bouton '`Supprimer`' qui efface l'entrée sélectionnée de la liste.
- un bouton '`Trier`' pour trier par ordre alphabétique la liste d'entrées.
- deux boutons '`Ok`' et '`Annuler`' pour quitter l'éditeur d'entrées.

Après avoir ajouté toutes les entrées ou les avoir changées, pressez le bouton '`Ok`' pour fermer la fenêtre.

### 15.2.4 Copie de champs

Dans le cas où vous avez besoin de plusieurs champs similaires, il vous est possible de copier un champ. Pour cela, sélectionnez le champ à dupliquer et cliquez sur le bouton '`Copier`' sous la liste de champ. Cette action ouvre une requête '`Copier le champ`' montrant les options du champ sélectionné. Modifiez certaines parties comme le nom puis cliquez sur '`Ok`' pour générer une copie du champ.

### 15.2.5 Modification de champs

Après la création d'un nouveau champ, il est toujours possible de modifier ses réglages. Pour cela, double-cliquez sur le nom du champ ce qui laissera apparaître la requête '`Modifier le champ`'. Cette requête est semblable à celle de création d'un champ (voir Section 15.2.1 [Creating fields], page 67) et permet de modifier certains paramètres. Les paramètres ne pouvant être modifiés, comme le type de champ, apparaissent grisés.

Les choses à prendre en compte lors d'un changement de champ sont les suivantes :

- Le type de champ ne peut être changé. Si vous devez changer le type d'un champ, il est préférable d'en créer un nouveau du type voulu et de copier les enregistrements du champ à remplacer vers le nouveau champ en entrant un programme BeeBase dans l'éditeur de requête (voir Section 14.2 [Query editor], page 59).
- Si vous modifiez la valeur par défaut d'un champ, seuls les nouveaux enregistrements auront cette nouvelle valeur d'initialisation.
- Certaines précautions sont à prendre pour le changement d'entrées des champs de type Choix. Les entrées sont uniquement utilisées pour visualiser le contenu du champ, mais en interne, ce sont des numéros qui sont stockés et utilisés comme index dans la liste d'entrées. Donc si vous modifiez l'ordre des entrées, vous ne modifiez pas le numéro en interne mais sa visualisation sous forme d'entrées ! Par conséquent, vous ne devriez pas changer l'ordre des entrées après avoir crée un champ de type Choix. Par contre l'ajout de nouvelles entrées à la fin de la liste ne pose pas ce genre de problème. Pour une plus grande flexibilité avec possibilité de changer l'ordre des entrées, utilisez plutôt un champ de type Texte associé à une '`liste déroulant`' (voir Section 15.3.3 [Field object editor], page 73).

- La table de référence d'un champ Référence ne peut pas être modifiée.

Si vous en avez terminé les modifications, pressez le bouton '`Ok`' pour fermer la requête.

### 15.2.6 Suppression de champs

Pour effacer un champ, cliquez sur son nom dans la liste des champs de l'éditeur de structure et appuyez sur le bouton '`Supprimer`' situé sous la liste. Une requête de confirmation apparaîtra et si vous confirmez en appuyant à nouveau sur '`Supprimer`', le champ sera définitivement effacé.

Un problème peut se poser lorsque le champ est utilisé ailleurs dans le projet. Dans ce cas, le champ ne peut être simplement effacé mais toutes les références à celui-ci doivent impérativement disparaître du programme. Si le champ à effacer est utilisé ailleurs, l'éditeur de programme apparaîtra et affichera la première occurrence de ce champ. Vous devrez alors modifier le programme de manière à ce qu'il ne contienne plus aucune référence à ce champ. Après chaque suppression d'une référence, vous passez à la suivante en appuyant sur le bouton '`Compiler`'. À tout moment vous pouvez annuler toute l'opération en appuyant sur '`Rétablir`' et en refermant l'éditeur de programmes.

### 15.2.7 Tri des champs

Le classement des champs dans la partie '`Champs`' de l'éditeur de structure peut se faire de plusieurs façons. Vous pouvez le faire à la main par glisser/déposer, ou utiliser le bouton '`Tri`' sous la liste qui permet un tri par ordre alphabétique ('`Tri 1`'), ou encore un tri suivant l'ordre de l'affichage de la liste ('`Tri 2`').

## 15.3 Gestion de l'affichage

Dans la partie '`Affichage`' de l'éditeur de structure, vous définissez l'agencement des éléments de la base de données dans l'interface utilisateur. Cette section comporte une partie choix, une partie liste et plusieurs boutons.

### 15.3.1 Champ d'affichage

Le champ d'affichage contient les éléments suivants :

- un élément de sélection avec deux réglages, '`Fiche de table`' et '`Fenêtre principale`'. Dans '`Fiche de table`' vous définissez l'agencement des champs de la table active dans l'interface utilisateur. Dans '`Fenêtre principale`' vous définissez l'arrangement des tables.
- une liste montrant la disposition actuelle de l'interface utilisateur. La liste se présente sous forme d'arborescence. Les éléments ayant une flèche à leur gauche sont des objets d'IHM composés pouvant être ouverts ou fermés en (double-) cliquant sur le symbole de la flèche. Un double clic sur l'élément lui-même ouvre une fenêtre permettant l'édition de ses réglages. Tous les objets d'IHM issus du même objet parent sont placés de manière identique (horizontalement ou verticalement). La représentation est déterminée par l'objet d'IIHM parent : les éléments des tableaux, onglets et fenêtres sont placés verticalement alors que les éléments des groupes sont placés en fonction des réglages définis par l'éditeur de groupe (voir Section 15.3.8 [Group editor], page 78).

- un bouton '+' ('Ajouter') pour ajouter la table sélectionnée ou le champ sélectionné (suivant ce qui est affiché dans la partie choix) à la liste. En général, les tables et les champs sont ajoutés à la liste automatiquement après leur création.
- un bouton '-' ('Supprimer') pour retirer de la liste ce qui est sélectionné. Si vous retirez une table, la totalité du formulaire de la table sera enlevé de l'interface utilisateur. De ce fait, vous ne pourrez plus voir la table dans l'interface projet et pourrez cacher des tables entières. Idem pour les champs : retirez un champ équivaut à le cacher de l'interface projet.
- deux boutons 'Haut' et 'Bas' pour déplacer l'élément sélectionné vers le haut ou vers le bas dans la liste.
- deux boutons 'Entrée' et 'Sortie' pour déplacer l'élément sélectionné vers un niveau hiérarchique plus élevé ou plus bas dans la liste.
- un bouton 'Texte' pour ajouter un objet texte dans la liste (voir Section 15.3.5 [Text editor], page 77).
- un bouton 'Image' pour ajouter un objet image (voir Section 15.3.6 [Image editor], page 78).
- un bouton 'Séparateur' pour insérer un espace entre les autres objets (voir Section 15.3.7 [Space editor], page 78).
- un bouton 'Balance' pour ajouter un objet balance dans la liste. Cet objet est très utile pour contrôler la taille des autres objets graphiques.
- un bouton 'Groupe' pour ajouter un objet groupe dans la liste. Avant d'appuyer sur ce bouton, vous pouvez sélectionner plusieurs éléments dans la liste pour les déplacer dans un nouveau groupe (voir Section 15.3.8 [Group editor], page 78).
- un bouton 'Registre' pour ajouter un groupe enregistré dans la liste. Comme pour l'objet groupe, il est possible de sélectionner plusieurs objets graphiques afin de les déplacer vers le nouveau groupe enregistré (voir Section 15.3.9 [Register group editor], page 79).
- un bouton 'Fenêtre' pour ajouter une nouvelle fenêtre dans la liste. Comme précédemment, il est possible de sélectionner plusieurs objets graphiques pour les déplacer dans la nouvelle fenêtre (voir Section 15.3.10 [Window editor], page 79).

Pour plus de précisions concernant les éléments graphiques ainsi que la façon de les utiliser, voir Section 5.8 [User interface], page 27.

### 15.3.2 Éditeur de table

Lors de l'ajout d'une table, un objet d'affichage par défaut est créé. Pour modifier ses paramètres, double-cliquer la table dans la liste 'Affichage' et la fenêtre 'Affichage de la table' apparaîtra. La fenêtre a trois sections séparées en onglet : 'Général', 'Onglet' et 'Enregistrement'.

La section 'Général' contient les éléments suivants :

- un champ numérique 'Poids' pour spécifier le poids d'un objet. La valeur de ce champ détermine la proportion de place occupée, relativement aux autres champs par la table lors du rendu final de la fenêtre.
- un champ 'Arrière-plan' avec une case à cocher 'Défaut' pour définir l'apparence de l'arrière-plan de la table. Si le champ 'Défaut' est coché, alors l'arrière-plan par défaut

sera affiché, autrement vous pouvez cliquer sur le champ '`Arrière-plan`' pour ouvrir une fenêtre de configuration de l'arrière-plan.

- une case à cocher '`A un onglet`' qui indique si un onglet doit être affiché en haut de la table. Décocher ce champ pour masquer l'onglet. Pour des précisions sur les onglets, voir Section 5.8.3 [Panels], page 28.

La section '`Onglet`' est activée seulement si le champ '`A un onglet`' est coché. Il contient les éléments suivants :

- un champ de texte '`Titre`' pour donner un titre à l'entête de l'onglet.
- un champ de texte '`Police de caractères`' avec un bouton déroulant pour choisir la police du titre. Si vous laissez le champ vide, la police par défaut sera utilisée.
- un champ '`Arrière-plan`' avec une case à cocher '`Défaut`' pour définir l'image de fond de l'entête du panel. Si le champ '`Défaut`' est coché, alors l'image de fond par défaut sera utilisée. Sinon vous pouvez cliquer sur '`Arrière-plan`' pour ouvrir une requête et en sélectionner une.
- un champ '`Num / Tous`'. S'il est coché, le numéro de l'enregistrement en cours et le nombre total d'enregistrements apparaîtront dans la partie droite de l'entête du panel.
- un champ '`Filtre`' qui, si coché, ajoute un bouton de filtrage à l'entête du panel et permet d'activer ou non le filtrage d'enregistrement de la table. S'il n'est pas coché, l'élément du menu '`Table - Modification du filtre`' sera aussi désactivé pour cette table et vous ne pourrez plus ajouter de filtre pour la table. Pour plus d'informations sur le filtrage des enregistrements, voir Section 10.1 [Record filter], page 49.
- un champ '`Flèches`' pour ajouter deux boutons flèches au masque de la table. Ces flèches permettent la navigation entre les enregistrements de la table. S'il n'est pas coché, vous ne pourrez pas naviguer dans les enregistrements et tous les sous-menus du menu '`Atteindre l'enregistrement`' ainsi que les éléments '`Rechercher`', '`Rechercher en avant`', et '`Rechercher en arrière`' du menu '`Table`' seront désactivés.

La section '`Enregistrement`' permet de spécifier une description d'enregistrement. La description d'enregistrement est utilisée à la création des entrées de journal et dans les listes déroulantes pour la sélection des enregistrements. Ce doit être un identifiant court mais unique. La section contient les éléments suivants :

- un champ Choix '`Description`' à définir sur '`Champs`' ou '`Calculé`'.
- Si '`Description`' est défini à '`Champs`' alors une liste de champs de la table est affichée. Vous pouvez choisir n'importe quel élément pour la description d'enregistrement. Si vous sélectionnez '`Numéro d'enregistrement`' alors le numéro d'un enregistrement sera ajouté à son affichage. Plusieurs éléments peuvent être sélectionnés et vous pouvez réarranger l'ordre des éléments par glisser-déplacer.
- Si '`Description`' est défini à '`Calculé`' alors le champ '`Calculer`' sera affiché pour indiquer une fonction qui calcule la description de l'enregistrement. La fonction peut soit retourner une expression ou une liste d'expressions (voir Section 16.29.14 [Compute record description], page 163).
- une case à cocher '`Utiliser des listes multi-colonnes`' pour indiquer l'arrangement préféré quand l'affichage est en liste .

Sous les sections séparées par des tabulations, il y a deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre. Si vous en avez terminé avec les modifications, pressez le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.3 Éditeur de champ

Lorsque vous ajoutez un champ à la liste, un objet graphique par défaut lui sera attribué. Pour modifier les réglages de cet objet, un double clic ouvre la fenêtre '`Affichage du champ`'. Le contenu de cette fenêtre dépend du type de champ. Les éléments suivants sont disponibles pour la plupart des types de champ :

- un champ texte '`Titre`' pour le titre qui sera affiché à côté de l'objet champ (ou à l'intérieur de l'objet si c'est un bouton). Si vous laissez cette partie vide, aucun titre ne sera affiché.
- une partie choix '`Position`' pour définir la position du titre (s'il y en a un) par rapport un champ objet. Vous avez le choix entre '`Gauche`', '`Droite`', '`Haut`' et '`Bas`'.
- un champ texte '`Raccourci`' pour la saisie d'une lettre qui sera utilisée conjointement avec la touche ***`Alt`*** (pour Linux et Windows) ou la touche ***`Amiga`*** (pour AmigaOS) pour activer l'objet.
- un champ '`Accueil`'. Si coché, l'objet Champ correspondant sera l'objet de démarrage du cycle de focus d'un nouvel enregistrement. C'est très pratique dans le cas où vous entrez vos données en commençant toujours par le même champ. Lorsque vous définissez un champ en tant qu'objet "home", tous les autres objets Champ de cette table perdront cette propriété.
- un champ '`Navigation cyclique`'. Si coché, l'objet fera partie de la chaîne de focus qui utilise la touche ***`Tab`*** pour faire défiler les objets. Ne cochez pas cette case si vous ne voulez pas inclure cet élément.
- un champ '`Lecture seule`' qui, si coché, passe le champ en lecture seule. Cela signifie que vous pouvez lire le contenu mais pas le modifier. À noter cependant que si vous ajoutez un bouton popup à l'objet, le contenu pourra être modifié par la sélection du popup.
- un champ choix '`Alignement`' pour fixer la représentation du contenu d'un objet. Vous avez le choix entre '`Centre`', '`Gauche`' et '`Droite`' pour respectivement afficher au centre, à gauche ou à droite.
- un champ numérique '`Poids`' pour définir le poids de l'objet. Cette valeur définie la place occupée par cet objet dans la représentation finale par rapport aux autres objets. Pour la plupart des types de champs, cela n'affectera que la taille horizontale parce que la plupart des objets ont une taille prédéfinie. Pour un champ de type Mémo, cela affectera également la taille verticale.
- un champ '`Police de caractères`' pour choisir la police utilisée pour l'affichage du contenu du champ. La police par défaut sera utilisée si vous laissez ce champ vide.
- un champ '`Arrière-plan`' avec une case à cocher '`Défaut`' pour définir la représentation du fond du champ. Si la case '`Défaut`' est cochée, un arrière-plan par défaut sera utilisé. Sinon vous cliquez sur '`Arrière-plan`' pour ouvrir une fenêtre de sélection pour l'arrière-plan.
- un champ édition '`Infobulle`' pour définir le texte qui sera affiché dans la bulle d'aide pour l'objet Champ.

- une zone '`Activé/désactivé`'. Si '`Toujours activé`' est sélectionné alors l'objet est toujours activé indépendamment de l'enregistrement affiché. '`Désactivé sur initial`' désactive l'objet pour l'enregistrement initial mais l'active dans tous les autres cas. Si '`Calculer activation`' est choisi alors une fonction pour calculer l'état d'activation de l'objet peut être spécifiée sur la droite. La fonction ne requiert aucun argument. Si elle renvoie NIL alors l'objet est désactivé, sinon il est activé. Dans le cas où le déclencheur est laissé vide ou ne peut pas être trouvé, l'objet est désactivé. Pour plus d'informations sur la façon d'utiliser ce déclencheur, voir Section 16.29.13 [Compute enabled function], page 163.
- deux boutons '`Ok`' et '`Annuler`' pour fermer et quitter la fenêtre.

Une fois les réglages effectués, appuyez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.4 Réglages liés au type

Parmi les éléments ci-dessus, ceux qui suivent sont spécifiques au type de champ :

- Pour les champs de type texte, il existe une page '`Extras`' qui contient :
  - un champ '`Afficher l'image`' qui, si coché, joint un champ image au champ texte pour afficher une image dont le nom provient du contenu du champ. Le champ image est placé au-dessus du champ texte. Si vous ne cochez pas ce champ, les réglages des éléments '`Titre du champ texte`', '`Masquer le champ texte`' et '`Taille`' sont superflus.
  - un champ '`Titre du champ texte`'. Si elle est cochée, le titre de l'objet Champ est placé à gauche de la partie texte pour ménager un espace plus important pour la partie image. Sinon, le titre sera placé à côté de la partie image.
  - un champ '`Masquer le champ texte`' pour dissimuler la partie texte dans l'interface utilisateur. Si elle est cochée, seule la partie image sera visible.
  - un champ '`Taille`' pour définir le redimensionnement du champ image. Si '`Redimensionnable`' est sélectionné, l'objet peut être redimensionné et pourrait devenir plus grand que la taille de l'image. '`Fixé`' adapte la taille de l'objet à la taille de l'image. Si la taille de l'image change en fonction des enregistrements, alors l'objet est également redimensionné à chaque fois. '`Défilable`' ajoute deux barres de déplacement à l'objet pour voir les images qui seraient plus grandes que la zone visible. Si '`Ajusté`' est sélectionné, la taille de l'image est redimensionnée de façon à s'adapter à la taille de l'objet affiché. '`Homothétique`' permet également le redimensionnement mais en préservant les proportions originales.
  - un champ '`Sélecteur de fichier`' qui, si coché, ajoute un bouton déroulant à droite du champ texte. Ce bouton permet d'afficher une requête pour sélectionner un nom de fichier.
  - un champ '`Sélecteur de polices de caractères`' pour ajouter une liste de sélection d'une police de caractères.
  - un champ '`Sélecteur de liste`'. Si coché, un bouton déroulant sera ajouté à la droite du champ texte permettant de choisir le texte parmi une liste. Le texte des entrées pour la liste déroulante est défini à droite du champ. Les éléments peuvent être soit '`Statiques`', soit '`Calculés`'. Si '`Statiques`' est choisi alors la liste de chaînes peut être saisie dans l'éditeur d'entrées qui s'ouvre après un clic

sur le bouton '`Modifier les entrées`'. Pour plus de renseignements sur l'éditeur d'entrées, consultez voir Section 15.2.3 [Label editor], page 68. Si '`Calculé`' est sélectionné alors un déclencheur peut être désigné dans le champ '`Compute`', il sera exécuté à chaque fois que le bouton déroulant sera cliqué. Cette fonction doit retourner un mémo contenant une entrée par ligne ou NIL pour une liste vide (voir Section 16.29.18 [Compute list-view labels], page 165). Seulement un des champs '`Sélecteur de fichier`', '`Sélecteur de polices de caractères`' et '`Sélecteur de liste`' peut être sélectionné à la fois.

- un champ '`Vue`' qui, si coché, rajoute un bouton à droite du champ texte permettant de lancer un programme de visualisation externe avec le contenu du champ comme argument. C'est très pratique dans le cas où vous stockez les noms de fichiers dans le champ et voudriez voir le contenu d'un fichier via un programme de visualisation externe. Ce programme de visualisation est défini dans le menu '`Préférences - Visionneuse externe`' (voir Section 7.1.3 [External viewer], page 35).

- pour les champs de type Choix, il existe une partie '`Type`' qui vous permet de choisir si le contenu du champ sera affiché par un '`Bouton cyclique`' ou bien par un groupe de '`Boutons radio`'. Si vous choisissez '`Bouton cyclique`' alors vous pouvez définir la position du titre parmi '`Gauche`', '`Droite`', '`Haut`', ou '`Bouton`'. Si vous choisissez '`Boutons radio`' alors les deux éléments '`Cadre`' et '`Horizontal`' permettent d'avoir une bordure autour des boutons et de définir l'aspect horizontal.
- pour les champs de type Réél il existe un champ Entier '`Num décimal`' dans lequel vous définissez le nombre de chiffres après la virgule qui seront affichés.
- pour les champs de type date, il y a un champ case à coche '`Calendrier`' qui ajoute un bouton pop-up à droite du champ de date pour ouvrir un calendrier.
- pour les champs de type Date il existe un champ choix '`Format`' pour définir l'aspect visuel. Vous avez le choix entre '`HH:MM:SS`', '`MM:SS`' et '`HH:MM`'. Si vous choisissez '`HH:MM`', les secondes ne seront pas affichées et les valeurs d'entrées seront considérées comme des minutes.
- pour les champs de type Référence il existe une page '`Extras`' qui comportent les éléments suivants :
  - une section '`Affichage`' où saisir la description d'un enregistrement qui spécifie le contenu d'un enregistrement référencé à afficher. Voir Section 15.3.2 [Table object editor], page 71, pour des précisions sur la configuration d'une description d'enregistrement. Notez que le champ de choix '`Description`' a une entrée '`Depuis la table`'. Si sélectionnée, la configuration de l'objet table est utilisée pour afficher l'enregistrement référencé.
  - une zone '`Déroulante`' dans laquelle vous pouvez spécifier quels enregistrements de la table référencée doivent être disponibles dans la liste déroulante du champ Référence et comment ils doivent être affichés. Si '`Enregistrements`' est positionné sur '`Tous`' alors tous les enregistrements sont disponibles. Si '`Enregistrements`' est positionné sur '`Correspondant au filtre`' alors seuls les enregistrements de la table référencée correspondant au filtre installé sont disponibles. Si '`Enregistrements`' est positionné sur '`Calculé`' alors une fonction pour calculer l'ensemble d'enregistrements peut être saisie dans le champ

'Calculer'. Cette fonction déclencheur doit retourner une liste qui est parcourue à la recherche d'enregistrements de la table référencée. Ces enregistrements identifiés sont affichés dans la liste déroulante. Les éléments qui ne sont pas des enregistrements sont ignorés. Voir Section 16.29.19 [Compute reference records], page 165, pour plus d'informations sur cette fonction. En plus de ces enregistrements spécifiés par le réglage 'Enregistrements', l'enregistrement initial et l'enregistrement courant de la table référencée peuvent être ajoutés à la liste déroulante en cochant respectivement les options 'Enregistrement initial' et 'Enregistrement courant'.

- un champ 'Affichage'. Si coché, la création de l'objet graphique représentant la référence sera un bouton. En cliquant sur celui-ci, vous verrez l'enregistrement référencé dans le masque de la table de référence. Pour cela, la fenêtre contenant la table de référence s'ouvrira au premier plan si nécessaire.
- un champ 'Affichage automatique' qui, si coché, ajoute un bouton à droite du champ référence pour une visualisation automatique ou non du champ. En l'activant, la table de référence sera mise à jour automatiquement avec l'enregistrement référencé à chaque fois que l'utilisateur passera à un autre enregistrement.
- un champ 'Filtre' qui, si coché, ajoute un bouton à droite du champ référence pour appliquer ou non un filtre de référence sur le champ. Voir Section 10.2 [Reference filter], page 50, pour plus d'informations sur les filtres de référence.

- pour les champs de type Virtuel, l'éditeur d'objet Champ contient une page 'Extras' avec les options suivantes :
  - une liste de choix 'Type' dans laquelle vous définissez comment le contenu du champ devra être affiché. Vous pouvez opter pour 'Booléen' qui utilise une zone à cocher pour afficher les valeurs booléennes, ou pour 'Texte' qui utilise une zone de texte pour afficher une ligne de texte (y compris la date, l'heure et les valeurs numériques), ou encore 'Liste' qui utilise une liste pour afficher une liste de lignes (exemple : le résultat d'une requête select-from-where).
  - en réglant l'élément 'Type' sur 'Texte', deux nouveaux champs seront disponibles : 'Alignement' pour définir comment sera présenté le contenu du champ, et 'Nombres décimaux' pour fixer le nombre de chiffres affichés après la virgule pour le contenu un champ de type Réel.
  - Si le champ 'Type' a été défini sur 'Liste' alors davantage de champs sont disponibles : 'Afficher les titres', 'Navigation cyclique', 'Sur double clic', 'Déclencheur sur glisser-déposer d'URL', et 'Déclencheur de tri-déplacement'. Si 'Afficher les titres' est coché, la premier ligne du contenu du champ est affiché en titre de la liste. Autrement aucun titre n'est affiché et la première ligne est ignorée. Si 'Navigation cyclique' est coché, l'objet fait partie du cycle de focus atteignable par la touche *Tab*. Décocher cela si vous voulez sauter cet élément. Dans 'Sur double clic' vous pouvez indiquer l'action de cet événement sur une entrée de la liste. 'Ne rien faire' ignore le double clic. 'Afficher l'enregistrement' affiche l'enregistrement correspondant à l'élément cliqué. Si vous sélectionnez 'Exécuter le déclencheur' vous pourrez préciser à droite le nom d'une fonction déclencheur appelée à chaque

double clic. Pour plus d'informations sur les déclencheurs et leurs arguments, voir Section 16.29.15 [Double click trigger], page 164. Dans '`Déclencheur sur glisser-déposer d'URL`', vous pouvez entrer le nom d'une fonction de déclenchement qui est appelée lorsque l'utilisateur fait un glisser-déposer d'une liste d'URL (p. ex. des noms de fichiers) sur la vue de liste. Cela peut être utilisé, par exemple, pour stocker des noms de fichiers externes dans un projet. Voir Section 16.29.16 [URL drop trigger], page 164, pour plus d'informations sur cette fonction de déclencheur, y compris les arguments qui lui sont transmis. Dans '`Déclencheur de tri-déplacement`', vous pouvez entrer le nom d'un déclencheur qui est appelé lorsque l'utilisateur fait un glisser-déposer d'un élément pour trier les éléments dans la vue liste. Cela peut être utilisé, par exemple, pour changer l'ordre des enregistrements dans une table ou pour réorganiser les lignes dans un champ mémo. Voir Section 16.29.17 [Sort drop trigger], page 165, pour plus d'informations sur ce déclencheur, y compris les arguments qui lui sont transmis.

- un champ '`Mise à jour automatique`'. Si coché, le champ Virtuel est recalculé automatiquement à chaque changement d'un élément dépendant. Cela inclut le changement d'enregistrement dans une table dépendante, la modification d'un champ dépendant, l'ajout ou la suppression d'un enregistrement dans une table dépendante, basculer le mode utilisateur / administrateur du projet ou recompiler le programme du projet. Les dépendances d'un champ Virtuel sont obtenues automatiquement depuis la fonction de calcul du champ Virtuel. Utilisez le menu '`Projet - Exporter la structure`' pour visualiser toutes les dépendances obtenues (il peut être nécessaire de recompiler le programme du projet afin de mettre à jour les dépendances lors de l'activation de cette fonction). Veuillez noter que le champ Virtuel est recalculé quel que soit l'enregistrement dans lequel un champ dépendant est modifié et sans tenir compte si la valeur du champ dépendant est modifiée depuis l'interface utilisateur ou au cours de l'exécution du programme. Toutefois, la mise à jour automatique des champs virtuels se déroule généralement seulement après que toutes les autres exécutions du programme sont terminées. Si '`Mise à jour automatique`' est décoché, la valeur du champ Virtuel n'est calculée que lorsque cela est explicitement demandé ou qu'elle est modifiée dans une fonction du programme.

- Voici les options supplémentaires pour les boutons :
  - un champ choix '`Type`' dans lequel vous choisissez entre '`Bouton texte`' et '`Bouton image`'.
  - si vous choisissez le type '`Bouton texte`' les options suivantes seront disponibles : '`Titre`', '`Police de caractères`', '`Arrière-plan`' et '`Défaut`' pour respectivement donner un titre au bouton, choisir sa police d'affichage et les réglages d'arrière-plan.
  - si vous choisissez le type '`Bouton image`' les options suivantes seront disponibles : '`Image`' pour choisir l'image et '`Taille`' pour définir sa taille.

### 15.3.5 Éditeur de texte

Lorsque vous ajoutez un objet de texte dans la liste d'affichage ou lorsque vous en modifiez un par double clic, la fenêtre '`Texte`' apparaîtra et proposera les options suivantes :

- un champ texte '`Titre`' pour définir le texte à afficher.

- un champ numérique '`Poids`' pour définir la largeur de l'objet de texte.
- un champ choix '`Police de caractères`' pour définir la police du texte. Si vous laissez le champ vide, la police par défaut sera utilisée.
- un champ '`Arrière-plan`' et un champ '`Défaut`' pour définir les réglages de l'arrière-plan de l'objet de texte.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.6 Éditeur d'image

L'éditeur d'images apparaît lorsque vous ajoutez un nouvel objet image ou lorsque vous double cliquez sur une image existante. Il contient les options suivantes :

- un champ '`Poids`' dans lequel vous définissez la largeur de l'objet image dans la représentation finale.
- un champ '`Image`' dans lequel vous choisissez l'image à afficher.
- un champ '`Taille`' dans lequel vous définissez le type de redimensionnement. Si vous choisissez '`Redimensionnable`' l'objet image sera redimensionnable, alors que si vous choisissez '`Fixé`', la taille sera fixe.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.7 Éditeur d'espacement

Après avoir ajouté un objet d'espacement à la liste d'affichage, vous pourrez modifier ses paramètres en double cliquant sur celui-ci, ce qui ouvrira la fenêtre '`Séparateur`' comportant les options suivantes :

- un champ '`Délimiteur`' qui, s'il est coché, affiche une barre horizontale ou verticale (en fonction de la représentation de l'objet parent) au milieu de l'objet d'espacement. Ceci est très utile pour séparer les différents éléments dans la fenêtre de représentation.
- un champ numérique '`Poids`' pour fixer la largeur de l'objet.
- deux champs '`Arrière-plan`' et '`Défaut`' pour les réglages de l'arrière-plan.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.8 Éditeur de groupe

Après avoir ajouté un objet de groupe à la liste d'affichage, vous pourrez modifier ses paramètres en double cliquant sur celui-ci, ce qui ouvrira la fenêtre '`Groupe`' comportant les options suivantes :

- un champ texte '`Titre`' pour donner un titre, centré au-dessus du groupe. Si vous laissez ce champ vide, aucun titre ne sera affiché.
- un champ numérique '`Poids`' pour fixer la largeur de l'objet.

- deux champs '`Arrière-plan`' et '`Défaut`' pour les réglages de l'arrière-plan.
- un champ '`Bordure`' qui, s'il est coché, affichera un cadre autour du groupe.
- un champ '`Horizontal`' qui, s'il est coché, affichera le groupe horizontalement et sera défini comme '`HGroup`' dans la liste d'affichage. Sinon, le groupe sera organisé verticalement et sera défini comme '`VGroup`' dans la liste d'affichage.
- un champ '`Espacement`' qui, si elle est sélectionnée, ajoute de l'espace entre les objets enfants du groupe. Sinon aucun espace supplémentaire ne sera ajouté entre les objets.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.9 Éditeur de registre

Vous pouvez modifier les réglages d'un objet groupe de registre en double-cliquant dessus, ce qui ouvrira la fenêtre '`Groupe registre`' proposant les options suivantes :

- un champ numérique '`Poids`' pour fixer la largeur de cet objet.
- un champ '`Entrée`' pour donner un nom à chaque page de registre. Il est recommandé d'avoir précisément le même nombre d'entrées que d'éléments présents dans le groupe de registre. Pour en savoir plus sur l'édition des entrées, voir Section 15.2.3 [Label editor], page 68.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

### 15.3.10 Éditeur de fenêtre

Vous pouvez modifier les réglages d'un objet fenêtre en double-cliquant dessus, ce qui ouvrira l'éditeur de fenêtre et proposera les options suivantes :

- un champ texte '`Titre`' dans lequel vous entrez le texte qui sera affiché dans la barre de titre de la fenêtre ainsi que dans le bouton de fenêtre optionnel.
- un champ texte '`Id`' (seulement sur Amiga) qui peut contenir jusqu'à quatre caractères correspondant à l'id d'une fenêtre MUI. Si vous ajoutez un id, alors la taille et la position de la fenêtre pourront être sauvegardés grâce à l'option "figer" de MUI. Notez que chaque id défini dans BeeBase devra être unique. Si par accident vous utilisez un id existant, les fenêtres partageront les mêmes réglages. Les identifiants commençant par un souligné '`_`' sont réservés à un usage interne à BeeBase. Si vous laissez le champ vide, aucun id ne sera défini et les dimensions de la fenêtre seront stockés dans le fichier du projet. La fenêtre principale a toujours comme id les quatre premiers caractères du nom du projet. Définir un id pour une fenêtre est utile dans le cas où l'édition d'un projet se fait sous différentes configurations, parce que les coordonnées de la fenêtre sont mémorisées individuellement pour chaque configuration.
- un champ '`Bouton`' qui, si coché, ajoute un bouton pour ouvrir la fenêtre dans la fenêtre parent. Si cette case n'est pas cochée, la fenêtre ne peut être ouverte qu'à partir d'un programme BeeBase, via la fonction SETWINDOWOPEN (voir Section 16.21.4 [SETWINDOWOPEN], page 148). Les options suivantes définissent l'affichage du bouton de fenêtre :

- un champ texte '`Raccourci`' dans lequel vous définissez le raccourci permettant d'activer le bouton de fenêtre.
- une zone '`Activé/Désactivé`'. Si '`Toujours activé`' est sélectionné, alors le bouton de fenêtre est toujours actif. '`Désactivé par défaut`' désactive le bouton s'il est dans une table et que celle-ci affiche son enregistrement initial, sinon le bouton est activé. Si '`Calcul actif`' est choisi alors une fonction pour calculer l'état d'activation du bouton de fenêtre peut être spécifiée sur la droite. La fonction ne requiert aucun argument. Si elle renvoie NIL alors le bouton est désactivé, sinon il est activé. Dans le cas où le déclencheur n'est pas renseigné ou ne peut pas être trouvé, le bouton de fenêtre est désactivé. Pour plus d'informations sur la façon d'utiliser ce déclencheur, voir Section 16.29.13 [Compute enabled function], page 163. De plus si la case '`Fermer la fenêtre si désactivé`' est cochée alors la fenêtre est automatiquement fermée lorsque le bouton est désactivé.
- une partie numérique '`Poids`' dans laquelle vous définissez la largeur du bouton de fenêtre.
- un champ '`Police de caractères`' pour définir la police du bouton de fenêtre. Si vous laissez ce champ vide, la police par défaut sera utilisée.
- deux champs '`Arrière-plan`' et '`Défaut`' pour définir les réglages de fond du bouton de fenêtre.
- deux boutons '`Ok`' et '`Annuler`' pour fermer la fenêtre.

Quand vous en avez terminé avec les réglages, cliquez sur le bouton '`Ok`' pour fermer la fenêtre.

## 15.4 Exporter une structure

Parfois, il peut être utile d'avoir une vue d'ensemble de toutes les tables et champs d'un projet, par exemple lorsque vous voulez écrire un programme BeeBase. Pour cela, sélectionnez l'option du menu '`Projet - Exporter la structure`', ce qui vous demandera un nom de fichier dans lequel sera stocké la liste des tables et des champs.

Vous obtiendrez en sortie le nom du projet suivi de toutes les tables qui y sont associées. Pour chaque table, vous aurez le détail de tous les champs avec leurs types et les fonctions optionnelles qu'elles lancent. Pour les champs de type Virtuel, vous aurez également les dépendances permettant la mise à jour automatique de la valeur des champs.

# 16 Programmation de BeeBase

Ce chapitre (le plus long de ce manuel) décrit le langage de programmation de BeeBase, ainsi que toutes les fonctions disponibles. En revanche, ce chapitre n'est pas conçu comme un guide général de programmation. Vous devez être familier avec les bases de la programmation et devez avoir déjà écrit quelques programmes simples (et fonctionnant correctement :-)).

## 16.1 Éditeur de programme

Pour saisir un programme dans un projet, ouvrez l'éditeur de programme en sélectionnant le menu '`Programme - Modifier`'. Si vous êtes configuré en code source interne (voir Section 7.2.7 [Program source], page 38), cela ouvre la fenêtre '`Édition de programme`' qui contient :

- un champ d'édition de texte où vous modifiez le programme du projet.
- un bouton '`Compiler & fermer`' pour compiler le programme et, en cas de réussite, sortir de l'éditeur de programme.
- un bouton '`Compiler`' pour compiler le programme. Si votre programme contient une erreur quelque part, alors une description de l'erreur est affichée dans le titre de la fenêtre et le curseur est positionné sur l'emplacement fautif.
- un bouton '`Rétablir`' qui annule tous les changements effectués depuis la dernière compilation réussie.

L'éditeur de programme est une fenêtre non modale, vous pouvez laisser la fenêtre ouverte et continuer à travailler avec le reste de l'application. Vous pouvez fermer l'éditeur à tout moment en cliquant sur le bouton de fermeture de sa fenêtre, si vous avez fait des changements depuis la dernière compilation réussie, alors une requête de sécurité vous demandera de confirmer la fermeture de la fenêtre.

Si vous avez configuré le menu '`Programme - Code Source`' sur '`Externe`' alors l'éditeur externe (voir Section 7.1.2 [External editor], page 34) est lancé avec le nom du fichier source externe lors de la sélection du menu '`Programme - Modifier`'. Cela vous permet d'éditer le programme dans votre éditeur de texte favori (voir Section 16.2 [External program source], page 81).

Vous pouvez également compiler le programme d'un projet sans ouvrir l'éditeur de programme en sélectionnant le menu '`Programme - Compiler`'. Cela peut être utile par exemple si vous avez fait des modifications dans un fichier d'inclusion externe et que vous désirez incorporer ces changements dans le programme du projet.

## 16.2 Code source externe

En choisissant le menu '`Programme - Code Source - Externe`' et en saisissant un nom de fichier, il est possible de rendre le code source du programme d'un projet accessible de l'extérieur. Cela vous permet de charger le code source du programme dans votre éditeur de texte favori pour programmer.

Si la compilation réussie, le programme compilé est intégré en tant que programme du projet et est utilisé lors de l'exécution des déclencheurs. Lorsque de la sauvegarde d'un projet, la dernière version du programme compilé avec succès est stockée dans le projet.

Du coup, après la sauvegarde et la fermeture d'un projet, le fichier source externe n'est plus nécessaire. Il est possible d'indiquer si les fichiers sources externes inutiles doivent être effacés automatiquement en cochant le menu '`Préférences - Nettoyer les sources externes`'.

L'état du menu '`Programme - Code source`' est mémorisé avec le projet, ainsi lors de la réouverture d'un projet utilisant la fonctionnalité du code source externe, le fichier source externe est recréé automatiquement. Si le fichier externe existe déjà et est différent de la version stockée dans le projet, une requête de sécurité demande confirmation avant d'écraser le fichier.

Sur Amiga il est possible d'envoyer la commande `compile` au port ARexx de BeeBase depuis votre éditeur. BeeBase lit alors le fichier source externe, le compile et retourne le statut de la compilation ainsi qu'un éventuel message d'erreur contenant le nom du fichier, la ligne, la colonne et une description de l'erreur. Cela permet de positionner le curseur à l'emplacement exact où l'erreur de compilation s'est produite. Voir Section 17.9 [ARexx compile], page 169, pour plus de détails sur les valeurs de retour et le format des erreurs.

## 16.3 Préprocesseur

Les programmes BeeBase sont analysés par un préprocesseur de manière similaire à ce qui est fait sur les programmes C. Cette section décrit comment utiliser les directives du préprocesseur.

Toutes les directives débutent par le symbole dièse `#` qui doit également être le premier caractère de la ligne, des caractères d'espacement ou de tabulation peuvent cependant apparaître après le `#`.

### 16.3.1 #define

`#define` *nom chaîne*

Définit un nouveau symbole ayant le nom et le contenu spécifiés. La *chaîne* peut être n'importe quel texte incluant des espaces et se terminant à la fin de la ligne. Dans le cas où *chaîne* ne pourrait pas tenir sur une seule ligne, il est possible d'utiliser les lignes suivantes en utilisant le caractère anti-slash \ à la fin de chacune des lignes (sauf la dernière). Si le symbole *nom* apparaît dans la suite du code source il est alors remplacé par la valeur *chaîne*.

Exemple : '`(PRINTF "X vaut %i" X)`' affiche '`X vaut 1`' (Les occurrences de *nom* dans les chaînes ne sont pas modifiées.)

Le remplacement des symboles définis est réalisé syntaxiquement, ce qui signifie que vous pouvez remplacer des symboles par n'importe quel texte, p. ex. vous pouvez définir votre propre syntaxe comme le montre l'exemple suivant :

```
#define BEGIN (
#define END )

BEGIN defun test ()
        ...
END
```

La chaîne de substitution d'une définition peut faire référence à d'autres symboles définis par la directive `#define` ce qui autorise des définitions imbriquées. Cependant un maximum de 16 définitions imbriquées est autorisé.

Voir aussi #undef, #ifdef, #ifndef.

### 16.3.2 #undef

`#undef` *nom*

Supprime la définition du symbole *nom*. Si *nom* n'est pas défini, rien ne se passe.

Voir aussi #define, #ifdef, #ifndef.

### 16.3.3 #include

`#include` *file*

Lit le contenu de *file* (une chaîne encadrée par des guillemets doubles). BeeBase recherche le fichier à charger dans le répertoire courant et dans le répertoire indiqué dans les préférences (voir Section 7.2.12 [Program include directory], page 39). Le contenu du fichier est alors traité par le compilateur comme s'il faisait partie du code source courant.

Un fichier externe peut inclure un ou plusieurs autres fichiers externes, avec cependant une limite maximum de 16 directives `#include` imbriquées. Pour éviter d'inclure plusieurs fois les fichiers il est possible d'utiliser la compilation conditionnelle.

Lors de l'externalisation de code sources, il convient d'être attentif : le débogage et la localisation des erreurs sont plus difficiles dans les fichiers externes. Il est préférable de n'externaliser que les codes sources bien testés et indépendants du projet.

### 16.3.4 #if

`#if` *expr-const*

Si le résultat de l'expression constante spécifiée est différent de NIL alors le texte présent jusqu'à la directive `#else`, `#elif` ou `#endif` correspondante est utilisé pour la compilation, sinon (c.-à-d. la valeur de l'expression est NIL) le texte jusqu'à la directive `#else`, `#elif` ou `#endif` correspondante est ignoré pour la compilation.

Pour le moment il n'est possible d'utiliser que TRUE et NIL comme expression constante.

Voir aussi #ifdef, #ifndef, #elif, #else, #endif.

### 16.3.5 #ifdef

`#ifdef` *nom*

Si le symbole spécifié a été défini par une directive `#define` alors le texte suivant jusqu'à la directive `#else`, `#elif` ou `#endif` correspondante est pris en compte dans la compilation, sinon il est ignoré.

Voir aussi #if, #ifndef, #elif, #else, #endif.

### 16.3.6 #ifndef

`#ifndef` *nom*

Si le symbole spécifié n'a pas été défini par une directive `#define` alors le texte suivant jusqu'à la directive `#else`, `#elif` ou `#endif` correspondante est pris en compte dans la compilation, sinon il est ignoré.

Voir aussi #if, #ifdef, #elif, #else, #endif.

### 16.3.7 #elif

`#elif` *expr-const*

N'importe quel nombre de directives `#elif` peut apparaître entre une directive `#if`, `#ifdef` ou `#ifndef` et sa directive `#else` ou `#endif` correspondante. Les lignes suivant la directive `#elif` sont prises en compte dans la compilation seulement si toutes les conditions suivantes sont remplies :

- L'expression constante dans la directive `#if` précédente a été évaluée à NIL, le symbole de la directive `#ifdef` précédente n'était pas défini ou le symbole de la directive `#ifndef` précédente était défini.
- L'expression constante de chacune des directives `#elif` intermédiaires a été évaluée à NIL.
- L'expression constante a été évaluée comme différente de NIL.

Si toutes les conditions précédentes sont remplies alors les directives `#elif` et `#else` suivantes sont ignorées jusqu'au `#endif` correspondant.

Voir aussi #if, #ifdef, #ifndef, #else, #endif.

### 16.3.8 #else

`#else`

Inverse le sens de la directive conditionnelle qui était active. Si la directive conditionnelle précédente indiquait que les lignes devaient être prises en compte, alors les lignes entre le `#else` et le `#endif` correspondant sont ignorées. Si la directive conditionnelle précédente indiquait que les lignes devaient être ignorées, alors les lignes suivantes sont prises en compte dans la compilation.

Les directives conditionnelles et les directives `#else` correspondantes peuvent être imbriquées jusqu'à un niveau maximum de 16 directives conditionnelles imbriquées.

Voir aussi #if, #ifdef, #ifndef, #elif, #endif.

### 16.3.9 #endif

`#endif`

Termine une section de lignes introduites par l'une des directives de compilation conditionnelle `#if`, `#ifdef` ou `#ifndef`. Chacune de ces directives doit avoir un `#endif` correspondant.

Voir aussi #if, #ifdef, #ifndef, #elif, #else.

## 16.4 Langage de programmation

BeeBase utilise un langage de programmation dont la syntaxe est proche du lisp. En fait, plusieurs constructions et fonctions ont été adoptées du lisp standard. Cependant BeeBase n'est pas totalement compatible avec du lisp standard. De nombreuses fonctions manquent (p. ex. les commandes de destruction) et le sens de certaines autres commandes est différent (p. ex. la commande `return`).

### 16.4.1 Pourquoi Lisp ?

L'avantage d'un langage dans le style de Lisp est qu'il est possible de programmer à la fois de manière fonctionnelle et impérative. Les langages fonctionnels sont de plus en plus populaires dans les applications mathématiques. Le concept de base des langages fonctionnels est l'utilisation des expressions. Les fonctions sont définies de "manière mathématique" et la récursivité est beaucoup utilisée.

Les langages de programmation impératifs (p. ex. C, Pascal, Modula) utilisent une description pas à pas des instructions exécutées par la machine. Ici, c'est l'état qui est le concept de base (p. ex. variables) et un programme calcule ses sorties en passant d'un état à un autre (p. ex. en assignant des valeurs à des variables).

Lisp combine ces deux techniques et par conséquent permet de choisir la façon dont on souhaite implémenter les choses. On utilise alors la technique qui correspond le mieux à un problème spécifique ou celle que l'on préfère.

### 16.4.2 Syntaxe Lisp

Une expression lisp est soit une constante, soit une variable, soit une application de fonction. Pour appeler une fonction, lisp utilise une notation préfixée. La fonction et ses paramètres sont encadrés par des parenthèses. Par exemple, pour ajouter deux valeurs `a` et `b`, on écrit :

```
(+ a b)
```

Toutes les expressions retournent une valeur, ainsi dans l'exemple précédent la somme de `a` et `b` est renvoyée. Les expressions peuvent être imbriquées, c'est à dire qu'il est possible de placer une expression en tant que sous expression d'une autre.

L'évaluation des fonctions est effectuée en utilisant le principe d'appel par valeur, cela signifie que les arguments sont évalués avant d'appeler la fonction.

Sauf si indication contraire, toutes les fonctions sont strictes, c'est à dire que *tous* les arguments de la fonction sont évalués avant que la fonction ne soit appelée. Certaines fonctions cependant sont non strictes, p. ex. `IF`, `AND` et `OR`. Ces fonctions peuvent ne pas évaluer tous leurs arguments.

### 16.4.3 Types de programmes

BeeBase distingue trois types de programmes. Le premier est le programme du projet. Dans un programme de ce type, il est possible de définir des fonctions et des variables globales. Les fonctions peuvent être utilisées comme déclencheurs pour les champs. La saisie d'un programme de projet est réalisée dans l'éditeur de programme (voir Section 16.1 [Program editor], page 81).

Le deuxième type correspond aux programmes de requête où il n'est possible de saisir que des expressions. Une expression est autorisée à référencer des variables globales et à appeler des fonctions définies dans le programme du projet. Cependant, il est impossible de définir de nouvelles fonctions ou variables globales dans un tel programme. On saisie un programme de requête par l'intermédiaire de l'éditeur de requête (voir Section 14.2 [Query editor], page 59).

Le troisième type représente les expressions de filtrage. Ici, il est uniquement possible de saisir des expressions contenant des appels à des fonctions BeeBase prédéfinies. Toutes les fonctions prédéfinies ne sont pas disponibles, seules celles n'ayant pas d'effet de bord,

p. ex. vous ne pouvez pas utiliser une fonction qui écrit des données dans un fichier. Les expressions de filtrage sont saisies dans la fenêtre de modification de filtrage (voir Section 10.1.2 [Changing filters], page 49).

### 16.4.4 Nomenclature

Dans un programme BeeBase, il est possible de définir des symboles tels que des fonctions et des variables globales ou locales. Les noms de ces symboles doivent suivre les conventions suivantes :

- Le premier caractère d'un nom doit être une lettre minuscule, cela différencie les symboles de programme des noms de table et de champs.
- Les caractères suivants peuvent être n'importe quelle lettre, chiffre ou caractère de soulignement. Les autres caractères, comme les umlauts allemands ne sont pas autorisés.

### 16.4.5 Accéder aux enregistrements

Pour accéder aux tables et aux champs dans un programme BeeBase, vous devez spécifier un chemin pour les atteindre. Un chemin est une liste de composants séparés par des points, où chaque composant est le nom d'une table ou d'un champ.

Les chemins peuvent être soit relatifs, soit absolus. Les chemins absolus sont reconnaissables à la première composante de leur nom qui est le nom de table, suivi par une liste de champs se terminant par le champ à atteindre. Par exemple le chemin absolu '`Personne.Nom`' accède au champ '`Nom`' de l'enregistrement courant de la table '`Personne`', de même le chemin absolu '`Personne.Pere.Nom`' accède au champ '`Nom`' de l'enregistrement référencé par le champ '`Pere`' (un champ de type Référence vers la table '`Personne`').

Les chemins relatifs possèdent déjà une table courante à laquelle ils se rapportent. Par exemple dans une expression de filtrage, la table courante est la table sur laquelle porte le filtre. Le chemin relatif pour un champ de la table courante est simplement son nom. Pour les champs qui ne sont pas directement accessibles via la table courante, mais indirectement via des champs de référence, les mêmes règles que pour les chemins absolus s'appliquent.

Il n'est pas toujours évident si un chemin donné est relatif ou absolu p. ex. supposons que nous soyons en train d'écrire une expression de filtrage pour la table '`Foo`' qui dispose d'un champ '`Bar`' mais qu'il existe également une table '`Bar`' ; dans ce cas saisir '`Bar`' est ambigu : de quoi parle t'on, de la table ou du champ ? Pour cette raison, les chemins sont tout d'abord traités comme étant relatifs. Si aucun champ n'est trouvé de cette manière, alors le chemin est traité comme étant absolu, dans notre exemple, le champ sera préféré.

Et si nous désirons accéder à la table dans notre exemple ? Dans ce cas, le chemin doit être considéré comme absolu, pour indiquer qu'un chemin est absolu il faut ajouter deux doubles points devant le chemin. Dans notre exemple il faudrait taper '`::Bar`' pour accéder à la table.

Pour mieux comprendre les chemins et leur sémantique, considérons dans l'exemple précédent que le champ '`Bar`' de la table '`Foo`' est une référence vers la table '`Bar`' qui contient un champ '`Nom`'. Maintenant, il est possible d'accéder au champ '`Nom`' en tapant '`Bar.Nom`' ou '`::Bar.Nom`'. Les deux expressions ont une signification différente. '`::Bar.Nom`' indique de prendre l'enregistrement courant de '`Bar`' et de retourner la valeur du champ '`Nom`' de cet enregistrement, tandis que '`Bar.Nom`' prend l'enregistrement courant de la table

'Foo', extrait du champ 'Bar', l'enregistrement référencé, et récupère la valeur de son champ 'Nom'.

Pour donner un exemple encore plus complet, considérons que la table 'Bar' dispose de deux enregistrements. L'un contient 'Ralph' et l'autre contient 'Steffen' dans le champ 'Nom'. Le premier enregistrement est l'enregistrement courant. De plus, la table 'Foo' dispose d'un enregistrement (que l'on considère comme étant le courant) dont le champ 'Bar' référence le deuxième enregistrement de la table 'Bar'. Maintenant '::Bar.Nom' est évalué en 'Ralph' et 'Bar.Nom' en 'Steffen'.

### 16.4.6 Types de données pour programmer

Le langage de programmation de BeeBase connaît les types de données suivants :

| Type | Description |
|---|---|
| Bool | toutes les expressions, les expressions différentes de NIL sont considérées comme TRUE. |
| Integer | entier long sur 32 bits, les valeurs de choix sont converties automatiquement en entiers |
| Real | double sur 64 bits |
| String | chaîne de caractères de longueur arbitraire |
| Memo | comme une chaîne, mais répartie sur plusieurs lignes |
| Date | valeur de date |
| Time | valeur horaire |
| Record | pointeur sur un enregistrement |
| File | descripteur de fichier pour la lecture / écriture |
| List | liste d'éléments, NIL est la liste vide. |

Tous les types de données supportent la valeur NIL.

### 16.4.7 Constantes

Le langage de programmation de BeeBase gère les expressions constantes qui peuvent être saisies en fonction de leur type :

| Type | Description |
|---|---|
| Integer | Les constantes entières dans l'intervalle -2147483648 2147483647 peuvent être indiquées telles quelles. Les valeurs débutant par 0 sont interprétées comme des |

nombres octaux, tandis que celles débutant par 0x
sont des nombres hexadécimaux.

Real Les constantes réelles dans l'intervalle -3.59e308
3.59e308 peuvent être spécifiées comme habituellement,
au format scientifique ou non. En l'absence de point
décimal le nombre peut être traité comme un entier
au lieu d'un réel.

String Les chaînes constantes sont une suite arbitraire de caractères
encadrée par des guillemets doubles. Par ex. "chaîne exemple".
Entre les guillemets doubles, tout caractère peut apparaître
hormis les caractères de contrôle et les retours à la ligne.
Cependant des codes d'échappement spéciaux
permettent de saisir ces caractères :

| | |
|---|---|
| \n | retour à la ligne (nl) |
| \t | tabulation horizontale (ht) |
| \v | tabulation verticale (vt) |
| \b | retour arrière (bs) |
| \r | retour chariot (cr) |
| \f | saut de page (ff) |
| \\ | antislash |
| \" | guillemet double |
| \e | code d'échappement 033 |
| *nnn* | caractère ayant le code octal *nnn* |
| \x*nn* | caractère ayant le code hexa *nn* |

Memo Comme pour les chaînes constantes (String).

Date Les valeurs constantes de date peuvent être spécifiées dans l'un des
formats 'JJ.MM.AAAA', 'MM/JJ/AAAA' ou 'AAAA-MM-JJ', où
'JJ', 'MM' et 'AAAA' sont des valeurs de deux ou
quatre chiffres représentant respectivement le jour, le mois et
l'année de la date.

Time Les valeurs constantes horaires peuvent être spécifiées au format
'HH:MM:SS', où 'HH' représente les heures, 'MM'
les minutes et 'SS' les secondes.

Pour quelques autres valeurs constantes prédéfinies, voir Section 16.25 [Pre-defined constants], page 154.

### 16.4.8 Convention typographique

Le reste de ce chapitre est consacré à la description de toutes les commandes et fonctions disponibles pour programmer dans BeeBase. La syntaxe suivante est utilisée pour la description des commandes :

- le texte entre crochets *[]* est optionnel. Si vous omettez le texte entre crochet, une valeur par défaut est utilisée.
- du texte séparé par une barre verticale `|` indique plusieurs options, p. ex. '`a | b`' signifie que vous pouvez spécifier soit '`a`' soit '`b`'.
- le texte écrit dans une police telle que *var* indique un paramètre qui est remplacé par une autre expression.
- les points de suspension ... indiquent que d'autres expressions peuvent suivre.
- tout autre texte est obligatoire.

## 16.5 Définition de commandes

Cette section liste les commandes permettant de définir des fonctions et des variables globales. Ces commandes ne sont disponibles que dans les programmes de projet.

### 16.5.1 DEFUN

`DEFUN` définit une fonction ayant le nom spécifié, une liste d'arguments passés à la fonction et une liste d'expressions à évaluer.

```
(DEFUN nom (liste-var) expr ...)
```

Le nom de la fonction doit commencer par une lettre minuscule, suivi d'autres lettres, chiffres ou caractères de soulignement (voir Section 16.4.4 [Name conventions], page 86).

Les paramètres *liste-var* spécifient les arguments de la fonction :

```
liste-var: var1 ...
```

où *var1* ... sont les noms des arguments. Ces noms doivent suivre les mêmes règles de nommage que ceux des fonctions.

Il est également possible de spécifier les type des arguments (voir Section 16.27 [Type specifiers], page 156).

La fonction exécute les expressions *expr*, ... une par une et retourne la valeur de la dernière. La fonction peut appeler d'autres fonctions y compris elle-même. L'appel d'une fonction définie par l'utilisateur est identique à l'appel d'une fonction prédéfinie.

Par exemple pour compter le nombre d'arguments d'une liste, vous pouvez définir la fonction suivante :

```
(DEFUN len (l)
    (IF (= l NIL)
        0
        (+ 1 (len (REST l)))
    )
)
```

Les fonctions définies par `DEFUN` sont affichées dans les listes déroulantes des fenêtres liées aux tables et aux champs (voir Section 15.1.1 [Creating tables], page 65) et Section 15.2.1 [Creating fields], page 67).

Cette commande n'est disponible que dans les programmes de projet.

Voir aussi DEFUN*, DEFVAR.

### 16.5.2 DEFUN*

`DEFUN*` est la version étoilée de `DEFUN` et a le même effet que `DEFUN` (voir Section 16.5.1 [DEFUN], page 89). La seule différence réside dans le fait que les fonctions définies avec `DEFUN*` ne sont pas affichées dans les listes déroulantes de création et de modification de table et de champ. Cependant, il est toujours possible d'entrer le nom de la fonction dans les champs texte correspondants.

Cette commande est uniquement disponible dans les programmes de projet.

Voir aussi DEFUN, DEFVAR.

### 16.5.3 DEFVAR

`(DEFVAR` *var* `[`*expr*`])`

Définit une variable globale ayant comme valeur initiale *expr* ou NIL si *expr* est absent. Les noms des variables doivent commencer par une lettre minuscule suivie de lettres, chiffres ou caractères souligné (voir Section 16.4.4 [Name conventions], page 86).

Il est également possible d'ajouter un spécificateur de type au nom de la variable (voir Section 16.27 [Type specifiers], page 156).

`DEFVAR` est uniquement disponible dans les programmes de projet. Toutes les commandes `DEFVAR` doivent être placées avant la définition de toutes les fonctions.

Après l'exécution d'un déclencheur (lorsque BeeBase rend la main à l'interface graphique), toutes les variables globales perdent leur contenu. Elles sont réinitialisées avec leur valeur initiale *expr* lors de la prochaine invocation d'un déclencheur. Si ce n'est pas voulu, il faut utiliser la commande `DEFVAR*` (voir Section 16.5.4 [DEFVAR*], page 90) qui permet de préserver la valeur des variables globales entre les appels de programme.

Il est conseillé de limiter (ou d'éviter complètement) l'utilisation des variables globales, car chacune doit être initialisée (et *expr* doit être évalué s'il est fourni) à chaque fois qu'un déclencheur est appelé de l'extérieur.

Exemple: '`(DEFVAR x 42)`' définit une variable globale '`x`' ayant la valeur 42.

Il existe quelques variables globales prédéfinies dans BeeBase (voir Section 16.24 [Predefined variables], page 154).

Voir aussi DEFVAR*, DEFUN, DEFUN*, LET.

### 16.5.4 DEFVAR*

`(DEFVAR*` *var* `[`*expr*`])`

`DEFVAR*` a le même effet que la commande `DEFVAR` (voir Section 16.5.3 [DEFVAR], page 90) sauf qu'une variable définie avec `DEFVAR*` ne perd pas sa valeur à la fin du programme.

Lors de la première invocation du programme, *var* est initialisée avec *expr* ou NIL si *expr* est omis. Les appels suivants du programme ne réévalueront pas *expr*, mais utiliseront la valeur de *var* de l'appel précédent. De cette manière, il est possible de transférer l'information d'un appel de programme à un autre sans avoir à stocker les données dans un

fichier externe ou une table de la base de données. Il faut cependant noter que toutes les variables globales définies avec `DEFVAR*` perdent leur valeur lorsque le programme du projet est recompilé. Pour conserver de manière permanente des informations, il est préférable d'utiliser un champ (éventuellement caché) d'une table.

Voir aussi DEFVAR, DEFUN, DEFUN*, LET.

## 16.6 Structures de contrôle

Cette section liste les fonctions contrôlant le programme, p. ex. les fonctions définissant les variables locales, les fonctions de boucle, les fonctions d'exécution conditionnelle, et d'autres.

### 16.6.1 PROGN

Pour évaluer plusieurs expressions l'une à la suite de l'autre, il est possible d'utiliser la construction `PROGN`.

`([`*expr ...*`])`

exécute *expr ...* une à une. Renvoie le résultat de la dernière expression (ou NIL si aucune expression n'a été spécifiée). En Lisp cette construction est connue comme `(PROGN` [*expr ...*]).

Exemple : '`(1 2 3 4)`' retourne 4.

Voir aussi PROG1.

### 16.6.2 PROG1

Une autre manière, hormis la fonction `PROGN`, pour évaluer plusieurs expressions séquentiellement est l'expression `PROG1`.

`(PROG1 [`*expr ...*`])`

exécute *expr ...* et renvoie la valeur de la première expression (ou NIL si aucune expression n'a été spécifiée).

Exemple : '`(PROG1 1 2 3 4)`' retourne 1.

Voir aussi PROGN.

### 16.6.3 LET

`LET` définit un nouveau bloc de variables locales. Ceci est utile, p. ex. pour définir les variables locales d'une fonction. La syntaxe est

`(LET (`*liste-var*`)` *expr ...*`)`

où *liste-var* est une liste de variables locales.

*liste-var* `:` *varspec ...*

*varspec* `: (`*var expr*`) |` *var*

Ici *var* est le nom d'une variable et doit commencer par une lettre minuscule, suivie d'autres lettres, chiffres ou caractères soulignés (voir Section 16.4.4 [Name conventions], page 86).

Dans le cas de l'utilisation de `(`*var expr*`)`, la nouvelle variable est initialisée par l'expression spécifiée. Dans le cas contraire, la nouvelle variable est positionnée à NIL.

Il est également possible d'ajouter des spécificateurs de type aux variables (voir Section 16.27 [Type specifiers], page 156).

Après l'initialisation de toutes les variables les expressions de la liste *expr ...* sont évaluées et la valeur de la dernière est renvoyée.

Par exemple, l'expression `LET` suivante

```
(LET ((x 0) y (z (+ x 1)))
    (+ x z)
)
```

renvoie 1.

Voir aussi DOTIMES, DOLIST, DO, DEFVAR.

### 16.6.4 SETQ

La fonction `SETQ` affecte des valeurs à des variables, des champs et des tables.

`(SETQ` *`lvalue1 expr ...`*`)`

Affecte à *lvalue1* la valeur de l'expression *expr*. Les points de suspension indiquent la possibilité d'affecter d'autres *parties gauches* (NDT: en anglais *lvalue* est l'abréviation de *left value*, c'est à dire littéralement *valeur de gauche* on utilise plutôt *partie gauche*). Une partie gauche est une variable, un champ d'une table ou une table. Dans le cas d'une variable, elle doit avoir été préalablement définie (p. ex. en utilisant l'expression `LET`).

Affecter la valeur à une table correspond à positionner son pointeur de programme ou d'interface : '`(SETQ` *`Table expr`*`)`' positionne le pointeur d'enregistrement du programme de *Table* à la valeur *expr*, '`(SETQ` *`Table* expr`*`)`' positionne son pointeur d'enregistrement de l'IHM et rafraîchit l'affichage. Pour plus d'informations à propos des pointeurs d'enregistrement de programme et d'IHM, voir Section 5.2 [Tables], page 20.

`SETQ` retourne la valeur de la dernière expression.

Exemple : '`(SETQ a 1 b 2)`' affecte 1 à la variable '`a`', 2 à la variable '`b`' et renvoie 2.

Voir aussi SETQ*, SETQLIST, LET, DEFVAR, Tables, Sémantique des expressions.

### 16.6.5 SETQ*

`SETQ*` est la version étoilée de `SETQ` (voir Section 16.6.4 [SETQ], page 92) et a les mêmes effets. La différence réside dans le fait que lors de l'affectation à un champ, `SETQ*` appelle le déclencheur de ce champ (voir Section 16.29.11 [Field trigger], page 162) au lieu de lui affecter directement la valeur. Dans le cas où aucun déclencheur ne serait associé à un champ, `SETQ*` se comporte comme `SETQ` et affecte simplement la valeur au champ.

Exemple : '`(SETQ* Table.Field 0)`' appelle le déclencheur de '`Table.Field`' avec un argument à 0.

Attention : Avec cette fonction il est possible d'écrire des boucles infinies, p. ex. si un déclencheur est défini pour un champ et que ce déclencheur utilise `SETQ*` pour affecter la valeur.

Voir aussi SETQ, SETQLIST*, LET, DEFVAR.

### 16.6.6 SETQLIST

`SETQLIST` est similaire à `SETQ` (voir Section 16.6.4 [SETQ], page 92) mais utiliser une liste pour définir plusieurs parties gauches pour les éléments de la liste.

```
(SETQLIST lvalue1 ... list-expr)
```

Sets *lvalue1* to the first element of *list-expr*. The dots indicate assignments for further lvalues to the corresponding elements in the list. The number of lvalues must be equal to the length of the list, otherwise program execution terminates with an error message.

Définit *lvalue1* sur le premier élément de *list-expr*. Les points indiquent les affectations pour d'autres parties gauches aux éléments de la liste correspondants. Le nombre de partie gauche doit être égal à la longueur de la liste, sinon l'exécution du programme se termine par un message d'erreur.

Les mêmes règles s'appliquent comme pour `SETQ` (voir Section 16.6.4 [SETQ], page 92). Par exemple, les deux versions suivantes des variables de réglage sont équivalentes :

```
(SETQLIST a b c (LIST 1 2 3))

(SETQ a 1 b 2 c 3)
```

`SETQLIST` retourne la valeur de *list-expr*.

Voir aussi SETQLIST*, SETQ.

### 16.6.7 SETQLIST*

`SETQLIST*` est la version étoilée de `SETQLIST` (voir Section 16.6.6 [SETQLIST], page 93) et a les mêmes effets.

La différence est que lors de l'attribution d'une valeur à un champ, `SETQLIST*` appelle le déclencheur de ce champ (voir Section 16.29.11 [Field trigger], page 162) au lieu d'attribuer directement la valeur. Dans le cas où aucun déclencheur n'a été spécifié pour un champ, `SETQLIST*` se comporte comme `SETQLIST` et attribue simplement la valeur au champ.

Voir aussi SETQLIST, SETQ*.

### 16.6.8 FUNCALL

`FUNCALL` est utilisé pour appeler une fonction avec des arguments.

```
(FUNCALL fonc-expr [expr ...])
```

Appelle la fonction *fonc-expr* avec les arguments spécifiés. L'expression *fonc-expr* peut être n'importe quelle expression dont la valeur est une fonction utilisateur ou prédéfinie, p. ex. une variable contenant la fonction à appeler. Si le nombre d'argument est incorrect, un message d'erreur est généré.

`FUNCALL` renvoie la valeur de retour de l'appel à la fonction ou NIL si *fonc-expr* est NIL.

Pour plus d'informations à propos des expressions fonctionnelles, voir Section 16.26 [Functional parameters], page 155.

Voir aussi APPLY.

### 16.6.9 APPLY

`APPLY` est utilisé pour appliquer une fonction à une liste d'arguments.

```
(APPLY fonc-expr [expr ...] liste-expr)
```

Applique la fonction *fonc-expr* à une liste créée en associant les arguments *expr* ... à *liste-expr*. En d'autres termes : appelle la fonction *fonc-expr* avec les arguments *expr* ... et *liste-expr* étendue à ses éléments de liste.

L'expression *fonc-expr* peut être n'importe quelle expression dont la valeur est une fonction utilisateur ou prédéfinie, p. ex. une variable contenant la fonction à appeler. Le dernier argument, *liste-expr*, doit être une liste valide ou NIL, dans le cas contraire un message d'erreur est généré. Si le nombre d'arguments n'est pas correct, une erreur survient.

`APPLY` renvoie la valeur de retour de l'appel de la fonction ou NIL si *fonc-expr* est NIL.

Pour plus d'informations à propos des expressions fonctionnelles, voir Section 16.26 [Functional parameters], page 155.

Exemple : '`(APPLY + 4 (LIST 1 2 3))`' renvoie 10.

Voir aussi FUNCALL.

### 16.6.10 IF

`IF` est un opérateur conditionnel.

`(IF` *expr1 expr2* `[`*expr3*`])`

L'expression *expr1* est évaluée. Si son résultat n'est pas NIL, alors la valeur de *expr2* est retournée, sinon c'est la valeur de *expr3* (ou NIL en cas d'absence).

Cette fonction n'est pas stricte, c'est à dire qu'une seule des expressions *expr2* ou *expr3* est évaluée.

Voir aussi CASE, COND.

### 16.6.11 CASE

`CASE` est similaire à l'instruction `switch` du langage C.

`(CASE` *expr* `[`*choix* `...])`

Ici *expr* est l'expression de sélection et *choix* ... des paires composées de :

*choix*`: (`*valeur* `[`*expr* `...])`

où *valeur* est une instruction ou une liste d'instructions et *expr* ... les expressions à exécuter si l'une des expressions du choix est remplie.

L'expression `CASE` évalue d'abord *expr*. Puis chaque paire de choix est testée pour vérifier si elle (ou l'une des expressions de la liste) correspond à l'expression évaluée. Si une expression de choix valide est trouvée, alors les expressions correspondantes sont exécutées et la valeur de la dernière expression est renvoyée. Si aucun choix ne convient, NIL est retourné.

Exemple : '`(CASE 1 ((2 3 4) 1) (1 2))`' retourne 2.

Voir aussi IF, COND.

### 16.6.12 COND

`COND` est, comme `IF`, un opérateur conditionnel.

`(COND [(`*test-expr* `[`*expr* `...]) ...])`

`COND` teste la première expression de chaque liste une à une. Pour la première qui ne renvoie pas NIL, les expressions associées *expr* ... sont évaluées et la valeur de la dernière est retournée.

Si toutes les expressions testées retournent NIL, alors la valeur de retour de COND sera également NIL.

### Exemple

```
(COND ((> 1 2) "1 > 2")
      ((= 1 2) "1 = 2")
      ((< 1 2) "1 < 2")
)
```

retourne "1 < 2".

Voir aussi IF, CASE.

## 16.6.13 DOTIMES

Pour des boucles simples, la commande `DOTIMES` est utilisable.

`(DOTIMES (`*nom entier-expr* `[`*résultat-expr ...*`]) [`*boucle-expr ...*`])`

Ici, *nom* est le nom d'une nouvelle variable qui sera utilisée dans la boucle. Le nom doit commencer par une lettre minuscule, suivie par d'autres lettres, chiffres ou caractères souligné (voir Section 16.4.4 [Name conventions], page 86).

Le nombre d'exécution de la boucle est indiqué par *entier-expr*. Dans *résultat-expr ...* il est possible de donner des expressions qui seront exécutées après la dernière boucle. *boucle-expr* correspond au corps de la boucle, c'est à dire les expressions qui sont évaluées dans chaque exécution de la boucle.

Avant d'exécuter la boucle, `DOTIMES` calcule la valeur de *entier-expr* pour déterminer le nombre d'exécution de la boucle. *entier-expr* n'est évaluée qu'une seule fois au début de la boucle et doit retourner une valeur entière. Ensuite, `DOTIMES` modifiera la valeur de la variable de boucle à chaque exécution de la boucle de 0 à *entier-expr*-1. Tout d'abord, la variable est initialisée à zéro puis comparée à la valeur de *expr* pour voir si elle est déjà supérieure ou égale. Si *entier-expr* est négative ou NIL ou si elle est supérieure ou égale à la valeur de *expr* alors la boucle se termine et les expressions de résultat sont évaluées. Dans le cas contraire, les expressions de la boucle sont évaluées et la variable est incrémentée de un, puis l'exécution revient au test d'arrêt et, éventuellement continue l'exécution de la boucle.

L'expression `DOTIMES` retourne la valeur de la dernière expression de résultat ou NIL si aucune expression de résultat n'a été donnée.

### Exemple

```
(DOTIMES (i 50 i) (PRINT i))
```

Affiche les nombres de 0 à 49 et retourne la valeur 50.

Voir aussi DOLIST, DO, FOR ALL, LET.

## 16.6.14 DOLIST

Pour boucler sur des listes, l'expression `DOLIST` peut être utilisée.

`(DOLIST (`*nom liste-expr* `[`*résultat-expr ...*`]) [`*boucle-expr ...*`])`

Ici, *nom* est le nom d'une nouvelle variable qui sera utilisée dans la boucle. Le nom doit commencer par une lettre minuscule, suivie par d'autres lettres, chiffres ou caractères souligné (voir Section 16.4.4 [Name conventions], page 86).

*liste-expr* spécifie la liste sur laquelle la boucle doit s'exécuter, *résultat-expr ...* sont des expressions qui seront évaluées après la dernière boucle, et *boucle-expr ...* correspond au corps de la boucle.

Avant d'exécuter la boucle, `DOLIST` calcule la valeur de *liste-expr*. Cette expression n'est évaluée qu'une seule fois au démarrage de la boucle et doit retourner une valeur de type liste. Ensuite, `DOTIMES` positionnera la variable de boucle à chacun des noeuds de la liste, un à chaque exécution de la boucle. Tout d'abord, la variable est initialisée au premier noeud de la liste. Si la liste est vide (NIL) alors la boucle se termine et les expressions de résultat sont évaluées. Dans le cas contraire, les expressions de la boucle sont évaluées et la variable est positionnée sur le noeud suivant de la liste, puis l'exécution revient au test d'arrêt et, éventuellement coutinue l'exécution de la boucle.

L'expression `DOLIST` retourne la valeur de la dernière expression de résultat ou NIL si aucune expression de résultat n'a été donnée.

### Exemple

```
(DOLIST (i (SELECT * FROM Comptes)) (PRINT i))
```

Affiche tous les enregistrements de la table '`Comptes`' et retourne NIL.

Voir aussi DOTIMES, DO, FOR ALL, LET.

## 16.6.15 DO

L'expression `DO` permet de programmer des boucles arbitraires.

```
(DO ([binding ...]) (terminaison-expr [résultat-expr ...]) [boucle-expr ...])
```

où *binding ...* est une déclaration de variable locale à la boucle, each of chacune ayant :

- un nouveau nom de variable (qui sera initialisé à NIL)
- une liste de la forme : `(`*nom init* [*pas*]`)` où *nom* est le nom de la nouvelle variable, *init* est la valeur initiale de cette variable et *pas* est l'expression de pas (NDT: c'est à dire que c'est elle qui modifie la valeur de la variable d'une exécution de boucle à une autre).

De plus *terminaison-expr* est l'expression du test d'arrêt, *résultat-expr ...* les expressions du résultat (NIL en cas d'absence) et *boucle-expr ...* correspond au corps de la boucle.

L'expression `DO` commence par initialiser toutes les variables locales avec leur valeur initiale, ensuite elle examine les expressions de terminaison. Si celles-ci retournent TRUE la boucle est arrêtée et les expressions de résultat sont évaluées, et la valeur de la dernière est retournée. Dans le cas contraire, le corps de la boucle `(`*boucle-expr ...*`)` est exécutée et chaque variable est mise à jour par la valeur de son expression de pas. Ensuite l'exécution teste la condition d'arrêt et ainsi de suite.

### Exemple

```
(DO ((i 0 (+ i 1))) ((>= i 5) i) (PRINT i))
```

Affiche les valeurs 0, 1, 2, 3 et 4 et renvoie la valeur 5. Bien sûr c'est une façon un peu compliquée d'écrire une simple boucle FOR. Par ailleurs il existe une version plus simple, l'expression `DOTIMES`.

Voir aussi DOTIMES, DOLIST, FOR ALL, LET.

### 16.6.16 FOR ALL

L'expression `FOR ALL` est utilisée pour boucler sur une liste d'enregistrements.

```
(FOR ALL table-liste [WHERE cond-expr] [ORDER BY ordre-liste] DO expr ...)
```

Ici, *table-liste* est une liste de tables séparées par des virgules, *cond-expr* une expression à valider pour chaque enregistrement, *ordre-liste* une liste d'expressions séparées par des virgules pour ordonner les enregistrements, et *expr ...* les expressions à exécuter pour chaque enregistrement.

`FOR ALL` commence par générer la liste de tous les enregistrements pour lesquels le corps de la boucle devra être exécuté. C'est réalisé de la même manière que dans l'expression `SELECT`. voir Section 16.20.12 [SELECT], page 146, pour plus d'informations sur la façon dont cette liste est générée. Pour chacun des éléments de cette liste, le corps de la boucle *expr ...* est exécuté.

Par exemple, faire la somme d'un champ d'une table peut être réalisé de la manière suivante :

```
(SETQ sum 0)
(FOR ALL Comptes DO
    (SETQ sum (+ sum Comptes.Solde))
)
```

L'expression `FOR ALL` renvoie NIL.

Voir aussi SELECT, DOTIMES, DOLIST, DO.

### 16.6.17 NEXT

`NEXT` peut être utilisé pour contrôler les boucles `DOTIMES`, `DOLIST`, `DO` et `FOR ALL`.

Un appel à `NEXT` dans le corps d'une boucle fera passer à la prochaine itération de la boucle. Cela peut être utilisé pour sauter des itérations de boucle sans intérêt, comme dans l'exemple suivant :

```
(FOR ALL Table DO
    (IF pas-intéressé-par-enregistrement-courant (NEXT))
    ...
)
```

Voir aussi EXIT, DOTIMES, DOLIST, DO, FOR ALL.

### 16.6.18 EXIT

`EXIT` peut être utilisé pour terminer une boucle.

```
(EXIT [expr ...])
```

À l'intérieur du corps d'une boucle, `EXIT` termine la boucle, exécute les éventuelles expressions *expr ...*, et renvoie la valeur de la dernière expression (ou NIL en cas d'absence) comme valeur de retour pour la boucle. Les éventuelles expressions de résultat de la boucle comme par exemple dans

```
(DOTIMES (x 10 résultat-expr ...) ...)
```

ne sont pas exécutées.

Il est par exemple possible d'utiliser la fonction `EXIT` pour terminer une boucle `FOR ALL` lorsque l'enregistrement recherché a été trouvé :

```
(FOR ALL Table DO
```

```
        (IF intéressé-par-enregistrement-courant (EXIT Table))
        ...
    )
```

Voir aussi NEXT, RETURN, HALT, DOTIMES, DOLIST, DO, FOR ALL.

### 16.6.19 RETURN

Lors de la définition d'une fonction, il est possible de rendre la main à l'appelant en utilisant la commande `RETURN`.

```
(RETURN [expr ...])
```

arrête l'exécution de la fonction, exécute les expressions *expr* ... optionnelles, et renvoie la valeur de la dernière (ou NIL en cas d'absence).

#### Exemple

```
(DEFUN find-record (nom)
    (FOR ALL Table DO
        (IF (= Nom nom) (RETURN Table))
    )
)
```

Cet exemple recherche un enregistrement dont le champ Nom correspond au nom spécifié. La fonction renvoie le premier enregistrement correspondant ou NIL si aucun enregistrement n'est trouvé.

Voir aussi HALT, EXIT.

### 16.6.20 HALT

`HALT` peut être utilisé pour terminer l'exécution du programme.

```
(HALT)
```

stoppe l'exécution du programme silencieusement.

Voir aussi ERROR, EXIT, RETURN.

### 16.6.21 ERROR

Pour stopper l'exécution du programme avec un message d'erreur, il est possible d'utiliser la fonction `ERROR`.

```
(ERROR fmt [arg ...])
```

stoppe l'exécution du programme et affiche une fenêtre avec un message d'erreur. Ce message est généré à partir de *fmt* et des arguments optionnels *arg* ... comme avec la fonction `SPRINTF` (voir Section 16.12.34 [SPRINTF], page 116).

Voir aussi HALT, SPRINTF.

## 16.7 Prédicats de type

Pour chaque type il existe un prédicat qui retourne TRUE si l'expression fournie est du type spécifié ou NIL dans le cas contraire. Ces prédicats sont :

| **Prédicat** | **Description** |
|---|---|

| | |
|---|---|
| (STRP *expr*) | TRUE si *expr* est de type chaîne, NIL sinon. |
| (MEMOP *expr*) | TRUE si *expr* est de type mémo, NIL sinon. |
| (INTP *expr*) | TRUE si *expr* est de type entier, NIL sinon. |
| (REALP *expr*) | TRUE si *expr* est de type réel, NIL sinon. |
| (DATEP *expr*) | TRUE si *expr* est de type date, NIL sinon. |
| (TIMEP *expr*) | TRUE si *expr* est de type heure, NIL sinon. |
| (NULL *expr*) | TRUE si *expr* est NIL (une liste vide), NIL sinon. |
| (CONSP *expr*) | TRUE si *expr* n'est pas la liste vide, NIL sinon. |
| (LISTP *expr*) | TRUE si *expr* est une liste (peut être NIL), NIL sinon. |
| (RECP *table expr*) | TRUE si *expr* pointe sur un enregistrement de la table spécifiée. Si *expr* est NIL, TRUE est renvoyé (enregistrement initial). Si *table* est NIL, vérification si *expr* est un pointeur d'enregistrement vers l'une des tables. |

## 16.8 Fonctions de conversion de type

Cette section liste les fonctions de conversion d'un type vers un autre.

### 16.8.1 STR

STR est utilisée pour convertir une expression vers une représentation en chaîne.

(STR *expr*)

convertit *expr* en une représentation chaîne. Le type de *expr* détermine la conversion :

| **Type** | **Chaîne retournée** |
|---|---|
| String | La chaîne elle-même. |
| Memo | Le texte complet du mémo dans une chaîne. |
| Integer | La représentation de la valeur entière. |
| Real | La représentation de la valeur réelle. Si *expr* est un champ de table, alors le nombre de décimales spécifié pour ce champ est utilisé, sinon 2 décimales sont produites. |

| | |
|---|---|
| Choice | Le label associé au choix. |
| Date | La représentation de la date en chaîne. |
| Time | La représentation de l'heure en chaîne. |
| Bool | La chaîne "TRUE" |
| NIL | La chaîne nil définie par l'utilisateur si *expr* est un champ ; la chaîne "NIL" sinon. |
| Record | La représentation du numéro d'enregistrement en chaîne. |
| Autres | La représentation de l'adresse interne du pointeur en chaîne. |

Voir aussi MEMO, SPRINTF.

### 16.8.2 MEMO

MEMO est utilisée pour convertir une expression en mémo.

(MEMO *expr*)

convertit *expr* en mémo. Elle traite l'expression de la même manière que la fonction STR (voir Section 16.8.1 [STR], page 99) mais retourne un mémo plutôt qu'une chaîne.

Voir aussi STR.

### 16.8.3 INT

INT est utilisée pour convertir une expression en entier.

(INT *expr*)

convertit *expr* en valeur entière. Les conversions possibles sont :

| **Type** | **Valeur retournée** |
|---|---|
| String | Si la chaîne complète représente une valeur entière valide, elle est convertie en entier. Une chaîne débutant par un 0 est interprétée comme un nombre octal, une débutant par 0x comme un nombre hexadécimal. Les espaces au début et à la fin sont ignorés.<br>Si la chaîne ne représente pas une valeur entière, NIL est retourné. |
| Memo | Idem que pour le type String. |
| Integer | La valeur elle-même. |
| Real | Si la valeur est comprise dans l'intervalle de valeurs autorisé pour les entiers, alors la valeur réelle est arrondie avant |

| | |
|---|---|
| | d'être retournée, sinon NIL est renvoyé. |
| Choice | Le numéro interne (à partir de 0) du label courant. |
| Date | Le nombre de jours depuis 01.01.0000. |
| Time | le nombre de secondes depuis 00:00:00. |
| Record | Le numéro d'enregistrement. |
| NIL | NIL |
| Autre | Un message d'erreur est généré et l'exécution du programme est arrêtée. |

Voir aussi REAL, ASC.

### 16.8.4 REAL

REAL est utilisée pour convertir une expression en une valeur de type réel.

(REAL *expr*)

convertit *expr* en un réel. L'expression est traitée de la même manière qu'avec la fonction INT (voir Section 16.8.3 [INT], page 100) mais renvoie une valeur de type réel au lieu d'un entier.

Voir aussi INT.

### 16.8.5 DATE

DATE est utilisée pour convertir une expression en une date.

(DATE *expr*)

convertit l'expression spécifiée en une valeur de type date. Les conversions possibles sont :

| **Type** | **Valeur retournée** |
|---|---|
| String | Si la totalité de la chaîne représente une date, alors la chaîne est convertie en une valeur de type date. Les espaces au début et à la fin sont ignorés.<br>Dans le cas contraire, NIL est renvoyé. |
| Memo | Idem que pour le type String. |
| Integer | Une date est générée en utilisant l'entier comme étant le nombre de jours depuis 01.01.0000. Si la valeur entière est trop grande (c.-à-d. la date serait supérieure au 31.12.9999) ou négative alors NIL est retourné. |
| Real | Idem que pour le type Integer. |

| | |
|---|---|
| Date | La date elle-même. |
| NIL | NIL |
| Autre | Un message d'erreur est généré et l'exécution du programme est arrêtée. |

Voir aussi DATEDMY.

### 16.8.6 TIME

TIME est utilisée pour convertir une expression en une heure.

(TIME *expr*)

convertit l'expression spécifiée en une valeur de type date. Les conversions possibles sont :

| **Type** | **Valeur retournée** |
|---|---|
| String | Si la totalité de la chaîne représente une heure, alors la chaîne est convertie en une valeur de type heure. Les espaces au début et à la fin sont ignorés.<br>Dans le cas contraire, NIL est renvoyé. |
| Memo | Idem que pour le type String. |
| Integer | Une heure est générée en utilisant l'entier comme étant le nombre de secondes depuis 00:00:00. |
| Real | Idem que pour le type Integer. |
| Time | L'heure elle-même. |
| NIL | NIL |
| Autre | Un message d'erreur est généré et l'exécution du programme arrêtée. |

## 16.9 Fonctions sur les booléens

Cette section liste les opérateurs booléens.

### 16.9.1 AND

AND vérifie que tous ses arguments sont à TRUE.

(AND [*expr* ...])

vérifie *expr* ... une à une jusqu'à ce qu'une expression retourne NIL. Si toutes les expressions sont différentes de NIL alors la valeur de la dernière expression est renvoyée, sinon NIL est retourné.

Cette fonction n'est pas stricte, ce qui signifie que ses arguments peuvent ne pas être évalués, p. ex. dans '`(AND NIL (+ 1 2))`' l'expression '`(+ 1 2)`' n'est pas évaluée puisqu'une valeur NIL a déjà été traitée, en revanche dans '`(AND (+ 1 2) NIL)`' l'expression '`(+ 1 2)`' sera bien évaluée.

Voir aussi OR, NOT.

### 16.9.2 OR

`OR` vérifie que tous les arguments sont NIL.

`(OR [`*expr ...*`])`

vérifie *expr ...* une à une jusqu'à ce qu'une expression retourne une valeur différente de NIL. Retourne la valeur de la première expression différente de NIL ou NIL si toutes les expressions sont NIL.

Cette fonction n'est pas stricte, ce qui signifie que ses arguments peuvent ne pas être évalués, p. ex. dans '`(OR TRUE (+ 1 2))`' l'expression '`(+ 1 2)`' n'est pas évaluée puisqu'une valeur différente de NIL a déjà été traitée, en revanche dans '`(OR (+ 1 2) TRUE)`' l'expression '`(+ 1 2)`' sera effectivement évaluée.

Voir aussi AND, NOT.

### 16.9.3 NOT

`NOT` est utilisée pour inverser la valeur d'une expression booléenne.

`(NOT` *expr*`)`

retourne TRUE si *expr* est NIL, NIL dans le cas contraire.

Voir aussi AND, OR.

## 16.10 Fonctions de comparaison

Cette section présente les fonctions de comparaison de valeur.

### 16.10.1 Opérateurs relationnels

Pour comparer deux valeurs dans un programme BeeBase, il faut utiliser

`(`*op expr1 expr2*`)`

où *op* est parmi {`=`, `<>`, `<`, `>`, `>=`, `<=`, `=*`, `<>*`, `<*`, `>*`, `>=*`, `<=*`}. L'étoile est utilisée pour les comparaisons spéciales (les chaînes sont comparées sans tenir compte de la casse, les enregistrements en utilisant l'ordre défini par l'utilisateur).

Le tableau suivant indique les règles utilisées pour comparer deux valeurs dans un programme BeeBase.

| **Type** | **Relation d'ordre** |
|---|---|
| `Entier` | `NIL < MIN_INT < ... < -1 < 0 < 1 < ... < MAX_INT` |
| `Réel` | `NIL < -HUGE_VAL < ... < -1.0 < 0.0 < 1.0 < ... < HUGE_VAL` |
| `Chaîne` | `NIL < "" < "Z" < "a" < "aa" < "b" < ... (sensible à la casse)` |
| | `NIL <* "" <* "a" <* "AA" <* "b" < ...` (insensible à la casse) |

```
Mémo            Identique aux chaînes

Date            NIL < 1.1.0000 < ... < 31.12.9999

Heure           NIL < 00:00:00 < ... < 596523:14:07

Booléen         NIL < TRUE

Enregistrement NIL < un_enregistrement               (les enregistrements ne sont pas
                                                                    comparables avec <)
                NIL <* rec1 <* rec2                (l'ordre définit par l'utilisateur)
```

Voir aussi CMP, CMP*, LIKE.

### 16.10.2 CMP

`CMP` retourne un entier représentant l'ordre de ses arguments.

`(CMP` *expr1 expr2*`)`

retourne une valeur inférieure à 0 si *expr1* est plus petite que *expr2*, 0 si *expr1* est égale à *expr2*, et une valeur supérieure à 0 si *expr1* est plus grand que *expr2*. Pour déterminer l'ordre, la relation d'ordre simple (non étoilée) est utilisée comme dans les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103).

Ne supposez pas que la valeur retournée sera toujours -1, 0 ou 1 !

Exemple : '`(CMP "Velo" "vELO")`' renvoie -1.

Voir aussi CMP*, Opérateurs relationnels.

### 16.10.3 CMP*

`CMP*` est la version étoilée de `CMP`. La différence est que `CMP*` utilise un ordre étendu comme défini dans les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103) où les chaînes sont comparées sans tenir compte de la casse et les enregistrements sont comparés en utilisant l'ordre défini par l'utilisateur.

Exemple : '`(CMP* "Velo" "vELO")`' renvoie 0.

Voir aussi CMP, Opérateurs relationnels.

### 16.10.4 MAX

`MAX` retourne l'argument ayant la plus grande valeur.

`(MAX [`*expr ...*`])`

Retourne la valeur la plus grande parmi les arguments *expr* . . . . Si aucun argument n'est passé, NIL est retouné. Pour identifier l'élément le plus grand, la relation d'ordre simple (non étoilée), comme pour les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103), est utilisée.

Voir aussi MAX*, MIN, Opérateurs relationnels.

### 16.10.5 MAX*

`MAX*` est la version étoilée de `MAX`. La différence est que `MAX*` utilise l'ordre étendu défini pour les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103) lorque les chaînes sont comparées sans tenir compte de la casse ou que les enregistrements sont comparés en utilisant l'ordre défini par l'utilisateur.

Exemple : '`(MAX* "x" "Y")`' retourne `"Y"`.

Voir aussi MAX, MIN*, Opérateurs relationnels.

### 16.10.6 MIN

`MIN` retourne l'argument ayant la plus petite valeur.

`(MIN [`*expr ...*`])`

Retourne la valeur la plus petite parmi les arguments *expr* .... Si aucun argument n'est passé, NIL est retouné. Pour identifier l'élément le plus petit, la relation d'ordre simple (non étoilée), comme pour les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103), est utilisée.

Voir aussi MIN*, MAX, Opérateurs relationnels.

### 16.10.7 MIN*

`MIN*` est la version étoilée de `MIN`. La différence est que `MIN*` utilise l'ordre étendu défini pour les opérateurs relationnels (voir Section 16.10.1 [Relational operators], page 103) lorque les chaînes sont comparées sans tenir compte de la casse ou que les enregistrements sont comparés en utilisant l'ordre défini par l'utilisateur.

Exemple : '`(MIN* "x" "Y")`' retourne `"x"`.

Voir aussi MIN, MAX*, Opérateurs relationnels.

## 16.11 Fonctions mathématiques

Ici quelques fonctions mathématiques sont listées.

### 16.11.1 Additionner des valeurs

Pour additionner des valeurs, utilisez

`(+ `*expr ...*`)`

Renvoie la somme des arguments *expr* .... Si l'un des arguments est NIL alors le résultat est NIL. Si les valeurs sont de type réel ou entier, alors la valeur du résultat sera de type réel ou entier.

Il est également possible d'ajouter des chaînes ou des mémos. Dans ce cas le résultat est la concaténation des chaînes ou des mémos.

Si *expr* est de type date et que le reste des arguments est de type entier / réel alors la somme des entiers / réels est interprétée comme un nombre de jours et est ajoutée à *expr*. Si la date résultante est hors limite (inférieure à 1.1.0000 ou supérieure à 31.12.9999) alors le résultat est NIL.

Si *expr* est de type heure et que le reste des arguments est de type entier, réel ou heure, alors la somme des entiers / réels (interprétée comme un nombre de secondes) ainsi que les valeurs horaires sont ajoutées à *expr*. Si l'heure résultante est hors limite (inférieur à 00:00:00 ou supérieure à 596523:14:07) alors le résultat est NIL.

### Exemples

| Expression | Valeur |
|---|---|
| `(+ 1 2 3)` | `6` |
| `(+ 5 1.0)` | `6.0` |
| `(+ "Hello" " " "world!")` | `"Hello world!"` |
| `(+ 28.11.1968 +365 -28 -9)` | `22.10.1969` |
| `(+ 07:30:00 3600)` | `08:30:00` |
| `(+ 03:00:00 23:59:59)` | `26:59:59` |

Voir aussi -, 1+, *, CONCAT, CONCAT2, ADDMONTH, ADDYEAR.

## 16.11.2 Soustraire des valeurs

Pour soustraire des valeurs, utilisez

`(- expr1 expr2 ...)`

Soustrait de *expr1* la somme de *expr2* .... Les mêmes règles que pour l'addition de valeurs (voir Section 16.11.1 [add], page 105) s'appliquent sauf que les chaînes et les mémos ne peuvent se soustraire.

`(- expr)` a une signification particulière : elle retourne la valeur négative de *expr* (entier ou réel), p. ex. '`(- (+ 1 2))`' retourne -3.

Voir aussi +, 1-.

## 16.11.3 1+

`1+` incrémente de un une expression entière, réelle ou une date.

`(1+ expr)`

Retourne la valeur de *expr* (entier, réel, date) augmentée de un. Si *expr* est NIL alors le résultat est également NIL.

Voir aussi +, 1-.

## 16.11.4 1-

`1-` décrémente de un une expression entière, réelle ou une date.

`(1- expr)`

Retourne la valeur de *expr* (entier, réel, date) diminuée de un. Si *expr* est NIL alors le résultat est également NIL.

Voir aussi -, 1+.

## 16.11.5 Multiplier des valeurs

Pour multiplier des valeurs entières / réelles, utilisez :

`(* expr ...)`

Retourne la multiplication des valeurs entières / réelles *expr* .... Si tous les arguments sont entiers alors un entier est retourné, sinon le résultat est de type réel.

Voir aussi +, /.

### 16.11.6 Diviser des valeurs

Pour diviser des valeurs entières / réelles, utilisez :

`(/ expr1 [expr2 ...])`

Divise *expr1* par le résultat de la multiplication du reste des arguments. Retourne une valeur réelle, NIL est renvoyé en cas de division par zéro.

Voir aussi *, DIV, MOD.

### 16.11.7 DIV

`DIV` permet d'effectuer des divisions entières.

`(DIV int1 int2)`

Retourne le résultat de la division entière de *int1* par *int2*. Par exemple, '`(DIV 5 3)`' renvoie 1.

Voir aussi /, MOD.

### 16.11.8 MOD

`MOD` permet de calculer le modulo.

`(MOD int1 int2)`

Retourne *int1* modulo *int2*. Par exemple, '`(MOD 5 3)`' renvoie 2.

Voir aussi DIV.

### 16.11.9 ABS

`ABS` calcule la valeur absolue d'une expression.

`(ABS expr)`

Retourne la valeur absolue de *expr* (entier ou réel). Si *expr* vaut NIL alors le résultat est NIL.

### 16.11.10 TRUNC

`TRUNC` tronque les décimales d'une valeur réelle.

`(TRUNC reel)`

Retourne le plus grand entier (sous forme réelle) qui n'est pas plus grand que le nombre réel spécifié. Si *reel* vaut NIL, le résultat est NIL.

Exemples : '`(TRUNC 26.1)`' renvoie 26, '`(TRUNC -1.2)`' renvoie -2.

Voir aussi ROUND.

### 16.11.11 ROUND

`ROUND` arrondit une valeur réelle.

`(ROUND reel décimales)`

Renvoie le nombre réel spécifié arrondi à *décimales* chiffres après la virgule. Si *reel* ou *décimales* sont NIL, le résultat est NIL.

Exemples : '`(ROUND 70.70859 2)`' renvoie 70.71, '`(ROUND 392.36 -1)`' renvoie 390.0.

Voir aussi TRUNC.

### 16.11.12 RANDOM

`RANDOM` permet de générer des nombres aléatoires.

`(RANDOM` *expr*`)`

Renvoie un nombre aléatoire. Au premier appel, le générateur de nombre aléatoire est initialisé avec une valeur générée à partir de l'heure courante. `RANDOM` génère un nombre dans l'intervalle 0 ... *expr*, en excluant la valeur *expr*. Le type de retour dépend de celui de *expr* : entier ou réel. Si *expr* est NIL, le résultat est NIL.

#### Exemples :

| **Exemple** | **Signification** |
|---|---|
| `(RANDOM 10)` | `retourne une valeur entre 0 et 9 compris,` |
| `(RANDOM 10.0)` | `retourne une valeur entre 0.0 et 9.99999... compris` |

### 16.11.13 POW

`POW` élève à une puissance un nombre.

`(POW` *x y*`)`

Retourne la valeur du nombre réel *x* élevé à la puissance du nombre réel *y*. Si *x* ou *y* est NIL, ou si *x* est négatif et *y* n'est pas un nombre entier, alors NIL est retourné.

Exemple : '`(POW 2 3)`' retourne 8.

Voir aussi SQRT, EXP.

### 16.11.14 SQRT

`SQRT` extrait la racine carré d'un nombre.

`(SQRT` *x*`)`

Retourner la racine carré d'une nombre réel *x*. Si *x* est NIL ou négatif alors NIL est retourné.

Voir aussi POW.

### 16.11.15 EXP

`EXP` calcule l'exponentielle d'un nombre.

`(EXP` *x*`)`

Retourne la valeur de la base du logarithme naturel élevé à la puissance du nombre réel *x*. Si *x* est NIL alors NIL est retouné.

Voir aussi POW, LOG.

### 16.11.16 LOG

`LOG` calcule le logarithme naturel d'un nombre.

`(LOG` *x*`)`

Retourne le logarithme naturel du nombre réel *x*. Si *x* est NIL ou n'est pas supérieur à 0 alors NIL est retourné.

Voir aussi EXP.

## 16.12 Fonctions sur les chaînes

Cette section concerne les fonctions ayant trait aux chaînes de caractères.

### 16.12.1 LEN

`LEN` calcule la longueur d'une chaîne.

`(LEN` *`str`*`)`

Retourne la longueur de la chaîne spécifiée ou NIL si *str* est NIL.

Voir aussi WORDS, LINES, MAXLEN.

### 16.12.2 LEFTSTR

`LEFTSTR` extrait une sous-chaîne d'une chaîne.

`(LEFTSTR` *`str long`*`)`

Retourne la partie gauche de la chaîne spécifiée avec au plus *long* caractères. Si *str* ou *long* sont NIL ou si *long* est négatif alors NIL est retourné.

Exemple : '`(LEFTSTR "Hello world!" 5)`' retourne "Hello".

Voir aussi RIGHTSTR, MIDSTR, WORD, LINE.

### 16.12.3 RIGHTSTR

`RIGHTSTR` extrait une sous-chaîne d'une chaîne.

`(RIGHTSTR` *`str long`*`)`

Retourne la partie droite de la chaîne spécifiée avec au plus *long* caractères. Si *str* ou *long* sont NIL ou si *long* est négatif alors NIL est retourné.

Exemple : '`(RIGHTSTR "Hello world!" 6)`' retourne "world!".

Voir aussi LEFTSTR, MIDSTR, WORD, LINE.

### 16.12.4 MIDSTR

`MIDSTR` extrait une sous-chaîne d'une chaîne.

`(MIDSTR` *`str pos long`*`)`

Retourne une partie de la chaîne spécifiée avec au plus *long* caractères. La sous-chaîne débute à la position *pos* (à partir de zéro). Si *long* est NIL alors toute la fin de la chaîne à partir de la position *pos* est retournée. Si *str* est NIL ou si *long* est négatif alors NIL est retourné. Si *pos* est hors limite, c.-à-d. négatif ou supérieur à la longueur de la chaîne, NIL est retourné.

Exemple : '`(MIDSTR "Hello world!" 3 5)`' retourne "lo wo".

Voir aussi LEFTSTR, RIGHTSTR, WORD, LINE, SETMIDSTR, INSMIDSTR.

### 16.12.5 SETMIDSTR

`SETMIDSTR` remplace une sous-chaîne d'une chaîne.

`(SETMIDSTR` *`str index set`*`)`

Retourne une copie de la chaîne *str* où la sous-chaîne débutant à *index* est remplacée par la chaîne *set*. La longueur de la chaîne retournée est supérieure ou égale à celle de *str*. Si l'un des arguments est NIL ou si *index* est hors limite, le résultat est NIL.

Exemple : '`(SETMIDSTR "Hello world!" 6 "Melanie!")`' retourne "Hello Melanie!".

Voir aussi INSMIDSTR, REPLACESTR.

### 16.12.6 INSMIDSTR

`INSMIDSTR` permet d'insérer une chaîne dans une autre.

```
(INSMIDSTR str index insert)
```

Retourne une copie de la chaîne *str* où la chaîne *insert* a été insérée à l'index spécifié. Si l'un des arguments est NIL ou si *index* est hors limite, le résultat est NIL.

Exemple : '`(INSMIDSTR "Hello world!" 6 "BeeBase-")`' retourne "Hello BeeBase-world!".

Voir aussi SETMIDSTR, REPLACESTR.

### 16.12.7 INDEXSTR

`INDEXSTR` cherche la première occurrence d'une chaîne dans une autre.

```
(INDEXSTR str substr)
```

Recherche la première occurrence de la chaîne *substr* dans *str* avec une comparaison sensible à la casse. Retourne un index (à partir de 0) de la première occurrence de la sous-chaîne dans *str* ou NIL si la sous-chaîne n'est pas présente. Si l'un des arguments est NIL alors le résultat est NIL.

Exemple : '`(INDEXSTR "Hello world!" "world")`' retourne 6.

Voir aussi INDEXSTR*, RINDEXSTR, RINDEXSTR*, INDEXBRK, INDEXBRK*.

### 16.12.8 INDEXSTR*

`INDEXSTR*` a le même effet que `INDEXSTR` (voir Section 16.12.7 [INDEXSTR], page 110) sauf que la comparaison de chaîne n'est pas sensible à la casse.

Voir aussi INDEXSTR, RINDEXSTR, RINDEXSTR*, INDEXBRK, INDEXBRK*.

### 16.12.9 INDEXBRK

`INDEXBRK` cherche la première occurrence d'un caractère dans une chaîne.

```
(INDEXBRK str listecar)
```

Recherche la première occurrence d'un des caractères contenus dans *listecar* dans la chaîne *str* avec une comparaison sensible à la casse. Retourne l'index (à partir de 0) du premier caractère trouvé dans *str* ou NIL si aucun caractère n'est trouvé. Si l'un des arguments est NIL, le résultat est NIL.

Exemple : '`(INDEXBRK "Hello world!" "aeiou")`' retourne 1.

Voir aussi INDEXBRK*, RINDEXBRK, RINDEXBRK*, INDEXSTR, INDEXSTR*.

### 16.12.10 INDEXBRK*

`INDEXBRK*` a le même effet que `INDEXBRK` (voir Section 16.12.9 [INDEXBRK], page 110) sauf que la comparaison de chaîne n'est pas sensible à la casse.

Voir aussi INDEXBRK, RINDEXBRK, RINDEXBRK*, INDEXSTR, INDEXSTR*.

### 16.12.11 RINDEXSTR

`RINDEXSTR` recherche la dernière occurrence d'une chaîne dans une autre.

```
(RINDEXSTR str substr)
```

Recherche la dernière occurrence de *substr* dans *str*, avec une comparaison sensible à la casse. Retourne l'index (à partir de 0) de la sous-chaîne dans *str* ou NIL si la sous-chaîne n'est pas présente. Si l'un des arguments est NIL, le résultat est NIL.

Exemple : '`(RINDEXSTR "Do itashimashite." "shi")`' returns 11.

Voir aussi RINDEXSTR*, INDEXSTR, INDEXSTR*, RINDEXBRK, RINDEXBRK*.

### 16.12.12 RINDEXSTR*

`RINDEXSTR*` a le même effet que `RINDEXSTR` (voir Section 16.12.11 [RINDEXSTR], page 111) sauf que la comparaison n'est pas sensible à la casse.

Voir aussi RINDEXSTR, INDEXSTR, INDEXSTR*, RINDEXBRK, RINDEXBRK*.

### 16.12.13 RINDEXBRK

`RINDEXBRK` recherche la dernière occurrence d'un caractère dans une chaîne.

```
(RINDEXBRK str listecar)
```

Recherche la dernière occurrence de l'un des caractères de *listecar* dans *str* avec une comparaison de chaîne sensible à la casse. Retourne l'index (à partir de 0) du dernier caractère trouvé dans *str* ou NIL si aucun caractère n'a été trouvé. Si l'un des arguments est NIL alors le résultat est NIL.

Exemple : '`(RINDEXBRK "Konnichiwa" "chk")`' retourne 6.

Voir aussi RINDEXBRK*, INDEXBRK, INDEXBRK*, RINDEXSTR, RINDEXSTR*.

### 16.12.14 RINDEXBRK*

`RINDEXBRK*` a le même effet que `RINDEXBRK` (voir Section 16.12.13 [RINDEXBRK], page 111) sauf que la comparaison n'est pas sensible à la casse.

Voir aussi RINDEXBRK, INDEXBRK, INDEXBRK*, RINDEXSTR, RINDEXSTR*.

### 16.12.15 REPLACESTR

`REPLACESTR` remplace des sous-chaînes par d'autres.

```
(REPLACESTR str [recherché1 remplaçant1 ...])
```

Remplace toutes les occurrences de *recherché1* dans *str* par *remplaçant1* et poursuit par le remplacement de sous-chaînes dans la nouvelle chaîne en utilisant la paire suivante de chaînes recherchée et remplaçante jusqu'au traitement complet des arguments. Noter que le nombre d'arguments doit être impair et que les arguments aux positions paires spécifient des chaînes recherchées suivies de la chaîne de remplacement correspondante. Du fait que le résultat d'un remplacement est utilisé lors du remplacement suivant, des remplacements multiples peuvent être effectués, ce qui doit être pris en compte lors de l'utilisation de cette fonction. Un ordre différent des arguments peut aider la résolution de conflits puisque les remplacements sont effectués de gauches à droite.

Si l'une des chaînes est NIL ou que l'une des chaînes de recherche est vide, le résultat est NIL.

Exemple : '`(REPLACESTR "black is white" "black" "white" "white "black")`' retourne "noir c'est noir".

Voir aussi REPLACESTR*, SETMIDSTR, INSMIDSTR, REMCHARS.

### 16.12.16 REPLACESTR*

`REPLACESTR*` a le même effet que `REPLACESTR` (voir Section 16.12.15 [REPLACESTR], page 111) sauf que la comparaison n'est pas sensible à la casse.

Voir aussi REPLACESTR, SETMIDSTR, INSMIDSTR, REMCHARS.

### 16.12.17 REMCHARS

`REMCHARS` supprime des caractères d'une chaîne.

`(REMCHARS` *str car-à-supprimer*`)`

Retourne une copie de *str* dans laquelle tous les caractères de *car-à-supprimer* ont été supprimés. Si *str* ou *car-à-supprimer* sont NIL, le résultat est NIL.

Exemple : '`(REMCHARS` *votre-chaîne* `" \t\n")`' supprime tous les espaces, tabulations et retour chariot de *votre-chaîne*.

Voir aussi REPLACESTR, TRIMSTR.

### 16.12.18 TRIMSTR

`TRIMSTR` supprime des caractères au début et à la fin d'une chaîne.

`(TRIMSTR` *str* `[`*début fin*`])`

Retourne une copie de *str* où les caractères de début et de fin ont été supprimés. Appelé avec un seul argument, les espaces, sauts de page, sauts de ligne, retours chariot et tabulations horizontales et verticales sont enlevés. Appelé avec trois arguments, *début* et *fin* spécifient respectivement les caractères à supprimer au début et à la fin de la chaîne. Il faut remarquer que `TRIMSTR` ne peut pas être appelé avec deux arguments.

Si l'un des arguments *str*, *début* ou *fin* est NIL alors le résultat est NIL.

Exemple : `(TRIMSTR " J'ai cassé le vélo de Selma. ")` retourne "J'ai cassé le vélo de Selma.", `(TRIMSTR "007 " "0" " \f\n\r\t\v")` retourne "7".

Voir aussi REMCHARS.

### 16.12.19 WORD

`WORD` retourne un mot contenu dans une chaîne.

`(WORD` *str num*`)`

Retourne le *num*ième mot (à partir de zéro) de la chaîne spécifiée. Les mots d'une chaîne sont des sous-chaînes non vides séparées par des caractères blancs (p. ex. espace, tabulation ou retour à la ligne).

Si *str* ou *num* sont NIL ou si *num* est hors limite, c'est à dire négatif, ou égal ou supérieur au nombre de mots, alors le résultat est NIL.

Exemple : '`(WORD "Du coup je prête mon vélo à Selma." 6)`' retourne "vélo".

Voir aussi WORDS, LINE, LEFTSTR, RIGHTSTR, MIDSTR.

### 16.12.20 WORDS

`WORDS` compte le nombre de mots contenus dans une chaîne.

```
(WORDS str)
```

Retourne le nombre de mots contenus dans la chaîne spécifiée ou NIL si *str* est NIL. Les mots sont des sous-chaînes non vides séparées par des caractères blancs (p. ex. espace, tabulation ou retour à la ligne).

Exemple : '`(WORDS "En fait, ce n'était pas vraiment mon vélo.")`' retourne 8.

Voir aussi WORD, LINES, LEN.

### 16.12.21 FIELD

`FIELD` extrait un champ d'une chaîne.

```
(FIELD str num [sep [quotes]])
```

Retourne le *num*ième champ (à partir de zéro) d'une chaîne. Les champs d'une chaîne sont des sous-chaînes séparées par exactement un caractère séparateur. Un champ peut être une chaîne vide. L'argument *sep* contient les caractères séparateurs de champ. Si *sep* est NIL ou absent alors un caractères d'espacement (e.g. espace, tab, ou saut de ligne) est utilisé. Si *quotes* est présent et n'est pas à NIL alors les champs peuvent être encadrés de guillemets double et peuvent contenir des caractères d'espacement. Les guillemets double sont supprimés pendant l'extration des champs.

Si *str* ou *num* sont NIL, ou si *num* est hors limite, c.-à-d., inférieur à 0, ou égale ou supérieur aux nombre de champs, alors NIL est retourné.

Exemple : '`(FIELD "My name is \"Darth Vader\"" 3 " " TRUE)`' retourne "Darth Vader".

Voir aussi FIELDS, WORD, LEFTSTR, RIGHTSTR, MIDSTR, STRTOLIST.

### 16.12.22 FIELDS

`FIELDS` compte le nombre de champs dans une chaîne.

```
(FIELDS str [sep [quotes]])
```

Retourne le nombre de champs dans une chaîne ou NIL si *str* est NIL. Les champs d'une chaîne sont des sous-chaînes séparées par exactement un caractère séparateur. Un champ peut être une chaîne vide. L'argument *sep* contient les caractères séparateurs de champ. Si *sep* est NIL ou absent alors un caractères d'espacement (e.g. espace, tab, ou saut de ligne) est utilisé. Si *quotes* est présent et n'est pas à NIL alors les champs peuvent être encadrés de guillemets double et peuvent contenir des caractères d'espacement. Les guillemets double sont supprimés pendant l'extration des champs.

Voir aussi FIELD, WORDS, LEN.

### 16.12.23 STRTOLIST

`STRTOLIST` convertit une chaîne en une liste de chaînes.

```
(STRTOLIST str [sep])
```

Crée une liste de chaînes en découpant la chaîne *str* au niveau des occurrences de la séquence de séparation *sep*. Si *sep* n'est pas spécifié alors le caractère tabulation `"\t"` est

utilisé. Si *sep* est la chaîne vide `""` alors la liste de tous les caractères contenus dans la chaîne est retournée.

Si *str* ou *sep* sont NIL alors NIL est retourné.

### Exemples

‘`(STRTOLIST "J'aime\tle\tJapon.")`’ retourne ( "J'aime" "le" "Japon." ).

‘`(STRTOLIST "Nom|Rue|Ville" "|")`’ retourne ( "Nom" "Rue" "Ville" ).

‘`(STRTOLIST "abc" "")`’ retourne ( "a" "b" "c" ).

Voir aussi MEMOTOLIST, LISTTOSTR.

## 16.12.24 LISTTOSTR

`LISTTOSTR` convertit une liste d'éléments en une chaîne.

```
(LISTTOSTR liste [sep])
```

Convertit la liste spécifiée d'éléments en une chaîne par concaténation des représentations textuelles de chaque élément de la liste séparées par la séquence *sep*. Si *sep* n'est pas spécifié alors le caractère tabulation `"\t"` est utilisé. Si *list* ou *sep* sont NIL alors NIL est retourné.

### Exemples

‘`(LISTTOSTR (LIST "Pierre a" 18 "ans"))`’ retourne "Pierre a\t18\tans".

‘`(LISTTOSTR (LIST "Nom" "Rue" "Ville") "|")`’ retourne "Nom|Rue|Ville".

Voir aussi LISTTOMEMO, CONCAT, CONCAT2, STRTOLIST.

## 16.12.25 CONCAT

`CONCAT` concatène des chaînes.

```
(CONCAT [str ...])
```

Retourne la concaténation de la liste des chaînes donnée avec des espaces entre chacune. Si l'une des chaînes est NIL ou si la liste est vide, le résultat est NIL.

Exemple : ‘`(CONCAT "Je" "pensais" "que" "c'était" "un" "vélo" "abandonné.")`’ retourne "Je pensais que c'était un vélo abandonné.".

Voir aussi CONCAT2, `+`, LISTTOSTR, COPYSTR, SPRINTF.

## 16.12.26 CONCAT2

`CONCAT2` concatène des chaînes.

```
(CONCAT2 insert [str ...])
```

Retourne la concaténation de la liste des chaînes donnée avec la chaîne *insert* entre chacune. Si *insert* ou l'une des chaînes est NIL ou si la liste est vide, le résultat est NIL.

Exemple : ‘`(CONCAT2 "! " "Mais" "ça" "ne" "l'était" "pas")`’ retourne "Mais! ça! ne! l'était! pas!".

Voir aussi CONCAT, `+`, LISTTOSTR, COPYSTR, SPRINTF.

### 16.12.27 COPYSTR

`COPYSTR` crée des copies d'une chaîne.

`(COPYSTR` *str num*`)`

Retourne une chaîne constituée de *num* fois la chaîne *str*. Si *str* est NIL, *num* est NIL ou inférieur à zéro, le résultat est NIL.

Exemple : '`(COPYSTR "+-" 5)`' retourne `"+-+-+-+-+-"`

Voir aussi CONCAT CONCAT2, `+`, SPRINTF.

### 16.12.28 SHA1SUM

`SHA1SUM` calcule la valeur de hachage SHA1 d'une chaîne.

`(SHA1SUM` *str*`)`

Retourne une chaîne contenant la valeur de hachage SHA1 de la chaîne spécifiée. Si *str* est NIL alors NIL est retourné.

Exemple : '`(SHA1SUM "flower, sun and beach")`' retourne "47b6c496493c512b40e042337c128d85ecf15ba4"

Voir aussi ADMINPASSWORD, démo `Users.bbs`.

### 16.12.29 UPPER

`UPPER` convertit une chaîne en majuscules.

`(UPPER` *str*`)`

Retourne une copie de la chaîne donnée où tous les caractères ont été convertis en majuscules. Si *str* est NIL, le résultat est NIL.

Exemple : '`(UPPER "Selma a trouvé une lettre attachée à mon vélo.")`' retourne "SELMA A TROUVÉ UNE LETTRE ATTACHÉE à MON VELO.".

Voir aussi LOWER.

### 16.12.30 LOWER

`LOWER` convertit en minuscules une chaîne.

`(LOWER` *str*`)`

Retourne une copie de la chaîne donnée où tous les caractères ont été convertis en minuscules. Si *str* est NIL, le résultat est NIL.

Exemple : '`(LOWER "La lettre était de Silke.")`' retourne "la lettre était de silke.".

Voir aussi UPPER.

### 16.12.31 ASC

`ASC` convertit un caractère en sa représentation interne entière.

`(ASC` *str*`)`

Retourne le code entier interne du premier caractère de *str*. Sur Windows, Mac OS et Linux il s'agit d'une représentation unicode. Sur Amiga, il s'agit du code entier 8 bits dans le jeu de caractères configuré dans le système. Si *str* est vide, le résultat est 0. Si *str* est NIL, le résultat est NIL.

Exemple : `(ASC "A")` retourne 65.

Voir aussi CHR, INT.

### 16.12.32 CHR

`CHR` convertit une valeur entière en caractère.

`(CHR` *entier*`)`

Retourne une chaîne contenant le caractère de code entier *entier*. Sur Windows, Mac OS et Linux *entier* est interprété comme un caractère unicode. Sur Amiga, *entier* est un entier 8 bits dans le jeu de caractères configuré dans le système. Si *entier* vaut 0, une chaîne vide est renvoyée. Si *entier* est NIL ou en dehors de l'intervalle des caractères valides, le résultat est NIL.

Exemple : '`(CHR 67)`' retourne `"C"`.

Voir aussi ASC, STR.

### 16.12.33 LIKE

`LIKE` compare des chaînes.

`(LIKE` *str1 str2*`)`

Retourne TRUE si *str1* correspond à *str2*, NIL sinon. La chaîne *str2* peut contenir les caractères jokers '?' (pour replacer exactement un caractère) et '*' (pour remplacer une suite quelconque de caractères). La comparaison est sensible à la casse.

Exemple : '`(LIKE "Silke est allé en France pendant un an." "*France*")`' retourne TRUE.

Voir aussi Fonctions de comparaison.

### 16.12.34 SPRINTF

`SPRINTF` formate une chaîne à partir de diverses données.

`(SPRINTF` *fmt* `[`*expr ...*`])`

`SPRINTF` prend une série d'arguments, les convertit en chaîne et retourne l'information formatée dans une seule chaîne. La chaîne *fmt* détermine exactement ce qui est écrit dans la chaîne retournée et peut contenir deux types d'éléments : des caractères ordinaires qui sont toujours copiés tels quels et des spécificateurs de conversion qui indique à `SPRINTF` de prendre en compte des arguments depuis la liste et de les formater. Les spécificateurs de conversion débutent toujours par un caractère '`%`'.

Les spécificateurs de conversion sont toujours de la forme :

`%[`*drapeaux*`][`*largeur*`][.`*precision*`]`*type*

où

- Le champ optionnel *drapeaux* contrôle la justification, le caractère de signe des valeurs numériques, le point décimal et les espaces de fin.
- Le champ optionnel *largeur* indique le nombre minimum de caractères à afficher (la largeur du champ), avec un bourrage par des blancs ou des zéros.
- Le champ optionnel *précision* spécifie soit le plus grand nombre de caractères à restituer pour les types chaîne, booléen, date et heure, soit le nombre de décimales à restituer après le séparateur décimal pour le type réel.
- Le champ *type* indique le type effectif de l'argument que `SPRINTF` va convertir : texte, entier, réel, etc.

Noter que le tous les champs ci-dessus sont optionnels à l'exception de *type*. Les tableaux suivants listent les options valides pour ces champs.

## Champ drapeaux

- : Le résultat est justifié à gauche, avec un bourrage par la droite avec des blancs. Par défaut lorsque '`-`' n'est pas spécifié, le résultat est justifié à droite avec un bourrage à gauche par des '`0`' ou des blancs.

`+` : Le résultat possède toujours un caractère représentant son signe s'il s'agit d'une conversion numérique.

0 : Pour les nombres justifiés à gauche, le remplissage est effectué avec des zéros au lieu d'espaces.

*espace* : Les nombres positifs débutent par un espace à la place d'un caractère '`+`', en revanche les valeurs négatives sont toujours préfixées d'un '`-`'.

## Champ largeur

*n* : *n* au minimum sont affichés. Si la conversion dispose de moins de *n* caractères, un remplissage par des espaces ou des zéros non-significatifs est effectué.

* : La valeur de la largeur est fournie dans la liste d'arguments en tant que valeur entière ou réelle, avant l'argument à convertir. Cette valeur est limitée à l'intervalle 0 à 999.

## Champ précision

.*n* : Pour les valeurs de type texte, booléen, date et heure, *n* est le nombre maximum de caractères de l'élément converti à restituer. Pour les conversions de valeurs réelles, *n* spécifie le nombre de chiffres après la virgule (conversions '`f`' et '`e`') ou le nombre de chiffres significatifs (conversion '`g`'). Dans les conversions d'entiers, ce champ est ignoré.

.*: La valeur de la précision est fournie dans la liste des arguments sous forme de valeur entière ou réelle, avant l'argument à convertir. Cette valeur est limitée à l'intervalle 0 à 999.

## Champ type

b : Convertit une valeur booléenne en `"TRUE"` ou `"NIL"`.

i : Convertit une valeur entière en une notation décimale signée.

o : Convertit une valeur entière en une notation octale non signée.

x : Convertit une valeur entière en une notation hexadécimale non signée, en utilisant les lettres minuscules '`abcdef`'.

X : Convertit une valeur entière en une notation hexadécimale non signée, en utilisant les lettres majuscules '`ABCDEF`'.

e : Convertit une valeur réelle en utilisant le format [-]d.ddde+dd (NDT : c.-à-d. la notation scientifique). Exactement un chiffre avant le point décimal, suivi

d'un '`e`', suivi d'un exposant. Le nombre de chiffres après le point décimal est déterminé par le champ précision ou est de 2 si aucune précision n'a été spécifiée. Le point décimal n'apparaît pas si la précision est de 0.

f : Convertit une valeur réelle en utilisant le format [-]ddd.ddd. Le nombre de chiffres après le point décimal est déterminé par le champ précision ou est de 2 si aucune précision n'est spécifiée. Le point décimal n'apparaît pas si la précision est de 0.

g : Convertit une valeur réelle en utilisant le style '`e`' ou '`f`' en fonction du nombre de chiffres représentant la valeur. Si la précision n'est pas précisée, 15 chiffres sont utilisés. Les zéros terminaux sont supprimés de la sortie, à l'exception de celui qui suit directement le point décimal.

s : Écrit la chaîne jusqu'à sa fin, ou jusqu'à ce que le nombre de caractères spécifié par le champ précision soit atteint.

d : Convertit une valeur de date.

t : Convertit une valeur horaire.

% : Écrit le caractère '`%`' sans qu'aucun argument ne soit converti.

`SPRINTF` retourne la chaîne formatée ou NIL si *fmt* est NIL.

## Exemples

| **Appel** | **Résultat** |
|---|---|
| `(SPRINTF "Hello")` | `"Hello"` |
| `(SPRINTF "%s" "Hello")` | `"Hello"` |
| `(SPRINTF "%10s" "Hello")` | `"     Hello"` |
| `(SPRINTF "%-10.10s" "Hello")` | `"Hello     "` |
| `(SPRINTF "%010.3s" "Hello")` | `"       Hel"` |
| `(SPRINTF "%-5.3b" TRUE)` | `"TRU  "` |
| `(SPRINTF "%i" 3)` | `"3"` |
| `(SPRINTF "%03i" 3)` | `"003"` |
| `(SPRINTF "%0- 5.3i" 3)` | `" 3   "` |
| `(SPRINTF "%f" 12)` | `"12.00"` |
| `(SPRINTF "%10e" 12.0)` | `"  1.20e+01"` |
| `(SPRINTF "%+-10.4f" 12.0)` | `"+12.0000  "` |
| `(SPRINTF "%10.5t" 12:30:00)` | `"     12:30"` |
| `(SPRINTF "%d" 28.11.1968)` | `"28.11.1968"` |
| `(SPRINTF "He%s %5.5s!" "llo" "world champion ship")` | `"Hello world!"` |

Voir aussi PRINTF, FPRINTF, STR, +, CONCAT, CONCAT2, COPYSTR.

## 16.13 Fonctions sur les mémos

Cette section traite des fonctions de manipulation des mémos.

### 16.13.1 LINE

`LINE` extrait une ligne d'un mémo.

`(LINE memo num)`

Retourne la *num*ième ligne (à partir de zéro) du mémo spécifié. La chaîne contenant la ligne ne contient pas de caractère de retour à la ligne à la fin. Si *memo* ou *num* sont NIL ou si *num* est hors limite, c.-à-d. négatif ou supérieur ou égal au nombre de lignes, le résultat est NIL.

Voir aussi LINES, WORD.

### 16.13.2 LINES

`LINES` retourne le nombre de lignes dans un mémo.

`(LINES memo)`

Retourne le nombre de lignes du mémo spécifié ou NIL si *memo* est NIL.

Voir aussi LINE, WORDS, LEN.

### 16.13.3 MEMOTOLIST

`MEMOTOLIST` convertit un mémo en une liste de chaînes.

`(MEMOTOLIST memo [expandstr])`

Convertit le mémo spécifié en liste. Si *memo* est NIL, alors le résultat est NIL, sinon, une liste est générée dans laquelle chaque élément contient une ligne du mémo.

Si *expandstr* est spécifié et n'est pas NIL alors la liste de chaînes retournée est ensuite traitée en appliquant `STRTOLIST` à chaque élément de la liste. Ce qui donne une liste de listes de chaînes.

#### Exemples

'`(MEMOTOLIST "Mon assurance\nprend en charge\nle vélo cassé.")`' retourne ( "Mon assurance" "prend en charge" "le vélo cassé." ).

'`(MEMOTOLIST "Voici\tun exemple\nmulti-colonnes." TRUE)`' retourne ( ( "Voici" "un exemple" ) ( "multi-colonnes." ) ).

Voir aussi STRTOLIST, LISTTOMEMO.

### 16.13.4 LISTTOMEMO

`LISTTOMEMO` convertit une liste en mémo.

`(LISTTOMEMO list)`

Convertit la liste spécifiée en mémo. Si *list* est NIL, alors le résultat est NIL, sinon un mémo est généré où chaque ligne est constituée de la conversion textuelle de l'élément correspondant de la liste. Si un élément de la liste est une sous-liste, alors `LISTTOSTR` (voir Section 16.12.24 [LISTTOSTR], page 114) lui est appliqué avant d'intégrer la chaîne résultat dans le mémo.

### Exemples

‘`(LISTTOMEMO (LIST "Silke" "me prête" "« mon » vélo" "jusqu'au" 01.09.1998)`’ retourne "Silke\nme prête\n« mon » vélo\njusqu'au\n01.09.1998".

‘`(LISTTOMEMO (LIST (LIST "Nom" "Date naissance") (LIST "Steffen" 28.11.1968)`’ retourne "Nom\tDate naissance\nSteffen\t28.11.1968".

Voir aussi LISTTOSTR, MEMOTOLIST.

## 16.13.5 FILLMEMO

`FILLMEMO` remplit un mémo avec les résultats d'expressions.

`(FILLMEMO` *memo*`)`

Crée une copie du mémo spécifié où toutes les sous-chaînes de la forme ‘`$(`*expr*`)`’ sont remplacées par leur résultat après leur évaluation.

Exemple : ‘`(FILLMEMO "(+ 1 1) donne $(+ 1 1).")`’ retourne "(+ 1 1) donne 2."

Le débogage et l'identification n'étant pas aisée ici, il est préférable de n'utiliser que des expressions simples dans un mémo.

Voir aussi FORMATMEMO, INDENTMEMO.

## 16.13.6 FORMATMEMO

`FORMATMEMO` formate un mémo.

`(FORMATMEMO` *memo longueur* `[`*bourrage* `[`*secmonoligne*`]])`

Formate *memo* de manière à ce qu'il devienne un mémo avec des lignes ayant *longueur* caractères au plus. Si *bourrage* n'est pas NIL des espaces de bourrage sont utilisés pour remplir les lignes jusqu'à *longueur* caractères. Le mémo est traité section par section, une section débute au premier caractère non blanc. Si *secmonoligne* est spécifié et différent de NIL alors tous les caractères jusqu'à la fin de cette ligne sont considérés comme faisant partie de la section. Sinon tous les caractères de cette ligne et des suivantes sont considérés dans la section jusqu'à rencontrer une ligne débutant par un caractère blanc. La section est ensuite formatée mot à mot, c'est à dire qu'autant de mots que possible sont mis dans une ligne. Les mots restants sont insérés dans la ligne suivante etc.

Voir aussi FILLMEMO, INDENTMEMO.

## 16.13.7 INDENTMEMO

`INDENTMEMO` indente un mémo en mettant des espaces à gauche.

`(INDENTMEMO` *memo indentation*`)`

Retourne une copie du mémo spécifié où chaque ligne est indentée par *indentation* espaces. Si *memo* ou *indentation* sont NIL, alors le résultat est NIL. Si *indentation* est négatif, une valeur de 0 est utilisée.

Voir aussi FILLMEMO, FORMATMEMO.

# 16.14 Fonctions sur les dates et heures

Cette section concerne les fonctions de manipulation de valeurs de type date et heure.

### 16.14.1 DAY

`DAY` extrait le jour d'une date.

```
(DAY date)
```

Renvoie un nombre entier représentant le jour de la date spécifiée. Si *date* est NIL alors NIL est retourné.

Voir aussi MONTH, YEAR, DATEDMY.

### 16.14.2 MONTH

`MONTH` extrait le mois d'une date.

```
(MONTH date)
```

Renvoie un nombre entier représentant le mois de la date spécifiée. Si *date* est NIL alors NIL est retourné.

Voir aussi DAY, YEAR, DATEDMY, MONTHDAYS.

### 16.14.3 YEAR

`YEAR` extrait l'année d'une date.

```
(YEAR date)
```

Renvoie un nombre entier représentant l'année de la date spécifiée. Si *date* est NIL alors NIL est retourné.

Voir aussi DAY, MONTH, DATEDMY, YEARDAYS.

### 16.14.4 DATEDMY

`DATEDMY` crée une date à partir du jour, du mois et de l'année.

```
(DATEDMY jour mois année)
```

Crée une date à partir des données du jour, du mois et de l'année. Si *jour*, *mois* ou *année* sont NIL, hors limite ou si le résultat de la date n'est pas valide, alors NIL est retourné.

Exemple : '`(DATEDMY 28 11 1968)`' donne le 28 novembre 1968.

Voir aussi DATE, TODAY, DAY, MONTH, YEAR.

### 16.14.5 MONTHDAYS

`MONTHDAYS` donne le nombre de jours d'un mois.

```
(MONTHDAYS mois année)
```

Renvoie le nombre de jours d'un mois dans une année. Si *mois* est NIL ou hors limite (inférieur à 1 ou supérieur à 12) alors NIL est retourné. Si *année* est NIL alors une année non bissextile est supposée pour calculer le nombre de jours. Si *année* est invalide (inférieure à 0 ou supérieure à 9999) alors NIL est retourné.

Exemple : '`(MONTHDAYS 2 2004)`' donne 29, '`(MONTHDAYS 2 NIL)`' donne 28.

Voir aussi YEARDAYS, MONTH.

### 16.14.6 YEARDAYS

`YEARDAYS` donne le nombre de jours d'une année.

```
(YEARDAYS année)
```

Renvoie le nombre de jours (Entier) contenus dans l'année spécifiée. Si *année* est NIL ou hors limite (inférieure à 0 ou supérieure à 9999) alors NIL est retourné.

Exemple : '`(YEARDAYS 2004)`' donne 366, '`(YEARDAYS 2005)`' donne 365.

Voir aussi MONTHDAYS, YEAR.

### 16.14.7 ADDMONTH

`ADDMONTH` ajoute un certain nombre de mois à une date.

```
(ADDMONTH date mois)
```

Renvoie une date où le nombre de mois spécifié a été ajouté à la date initiale. Les valeurs négatives pour *mois* soustraient les mois. Si *date* ou *mois* sont NIL ou si le résultat de la date est invalide, alors NIL est retourné.

`ADDMONTH` gère le débordement ou l'épuisement du champ mois en ajustant l'année en conséquence. Dans le cas où le champ jour dépasserait le nombre maximum de jours pour le mois résultant, il est décrémenté au nombre maximum de jours autorisé pour ce mois.

Exemple : '`(ADDMONTH 30.01.2004 1)`' donne 29.02.2004, '`(ADDMONTH 30.01.2004 -1)`' donne 30.12.2003.

Voir aussi ADDYEAR, `+`.

### 16.14.8 ADDYEAR

`ADDYEAR` ajoute un certain nombre d'années à une date.

```
(ADDYEAR date années)
```

Renvoie une date où le nombre d'années spécifié a été ajouté à la date initiale. Les valeurs négatives pour *années* soustraient des années. Si *date* ou *années* sont NIL ou si le résultat de la date est invalide, alors NIL est retourné.

`ADDYEAR` décrémente le jour de 1 au cas où la *date* représenterait le 29 février et si l'année résultante n'est pas bissextile.

Exemple : '`(ADDYEAR 29.02.2004 1)`' donne 28.02.2005, '`(ADDMONTH 04.02.2004 -1962)`' donne 04.02.0042.

Voir aussi ADDMONTH, `+`.

### 16.14.9 TODAY

`TODAY` renvoie la date du jour.

```
(TODAY)
```

Renvoie la date du jour en tant que valeur date.

Voir aussi NOW, DATEDMY.

### 16.14.10 NOW

`NOW` renvoie l'heure courante.

`(NOW)`

Renvoie l'heure courante avec une valeur de type Heure.

Voir aussi TODAY.

## 16.15 Liste des fonctions

Cette section énumère les fonctions pour le traitement des listes.

### 16.15.1 CONS

`CONS` construit une paire d'expressions.

`(CONS` *elem list*`)`

Construit une nouvelle liste. Le premier élément de la nouvelle liste est *elem*, le reste sont les éléments de *list* (qui devraient être une liste ou NIL). La liste *list* n'est pas copiée, seul un pointeur est utilisé pour la référencer !

Exemple : '`(CONS 1 (CONS 2 NIL))`' donne ( 1 2 ).

Les éléments d'une liste peuvent être de n'importe quel type, par exemple il est également possible d'avoir une liste de listes (par exemple voir voir Section 16.20.12 [SELECT], page 146). Le constructeur `CONS` peut également être employé pour établir des paires d'éléments, par exemple '`(CONS 1 2)`' est une paire avec les deux nombres entiers 1 et 2.

Voir aussi LIST, FIRST, REST.

### 16.15.2 LIST

`LIST` produit une liste en dehors de ses arguments.

`(LIST [`*elem ...*`])`

Prend les arguments *elem ...* pour produire une liste. Ce qui équivaut à appeler `(CONS` *elem* `(CONS ... NIL))`. Notez que NIL reste seul pour une liste vide.

Voir aussi CONS, LENGTH.

### 16.15.3 LENGTH

`LENGTH` détermine la longueur d'une liste.

`(LENGTH` *list*`)`

renvoie la longueur des données de la liste.

Exemple : '`(LENGTH (LIST "a" 2 42 3))`' donne 4.

Voir aussi LIST.

### 16.15.4 FIRST

`FIRST` extrait le premier élément dans une liste.

`(FIRST` *list*`)`

renvoie le premier élément d'une liste. Si *list* est vide (NIL) alors NIL est retourné.

Voir aussi REST, LAST, NTH, CONS.

### 16.15.5 REST

`REST` renvoie la sous-liste après le premier élément d'une liste.

`(REST` *list*`)`

renvoie le reste des données d'une liste (la liste sans premier élément). Si *list* est vide (NIL) alors NIL est retourné.

Exemple : '`(REST (LIST 1 2 3))`' donne ( 2 3 ).

Voir aussi FIRST, CONS.

### 16.15.6 LAST

`LAST` extrait le dernier élément dans une liste.

`(LAST` *list*`)`

Renvoie le dernier élément d'une liste ou NIL si *list* est NIL.

Voir aussi FIRST, NTH.

### 16.15.7 NTH

`NTH` extrait le N ième élément d'une liste.

`(NTH` *n list*`)`

Renvoie *n*-nième élément des données de la liste (commençant par 0) ou NIL si l'élément n'existe pas.

Voir aussi FIRST, LAST, REPLACENTH.

### 16.15.8 REPLACENTH

`REPLACENTH` remplace le N ième élément d'une liste.

`(REPLACENTH` *n elem list*`)`

Retourne une liste où le *n*-ième élément de la liste (commençant par 0) a été remplacé par *elem*.

Si *n* est NIL ou si le *n*-ième élément n'existe pas, retourne NIL.

Voir aussi NTH, REPLACENTH*, MOVENTH, REMOVENTH.

### 16.15.9 REPLACENTH*

`REPLACENTH*` est la version étoilée de `REPLACENTH`. La seule différence avec `REPLACENTH` est que `REPLACENTH*` retourne la liste initiale, si le *n*-ième élément n'existe pas.

`REPLACENTH*` peut être implémenter comme ceci

```
(DEFUN REPLACENTH* (n e l)
    (LET ((r (REPLACENTH n e l)))
        (IF r r l)
    )
)
```

Voir aussi NTH, REPLACENTH, démo `FileList.bbs`.

### 16.15.10 MOVENTH

`MOVENTH` déplace le N-ième élément d'une liste à une nouvelle position.

`(MOVENTH` *`n m list`*`)`

Renvoie une nouvelle liste où le *n*-ième élément de la liste (commençant par 0) a été déplacé vers la *m*-ième position.

Si *n* ou *m* est NIL ou si le *n*-ième ou le *m*-ième élément n'existe pas, retourne NIL.

Exemple : '`(MOVENTH 0 1 (list 1 2 3))`' retourne ( 2 1 3 ).

Voir aussi NTH, MOVENTH*, REPLACENTH, REMOVENTH.

### 16.15.11 MOVENTH*

`MOVENTH*` est la version étoilée de `MOVENTH`. La seule différence avec `MOVENTH` est que `MOVENTH*` retourne la liste initiale, si le *n*-ième or le *m*-ième élément n'existe pas.

`MOVENTH*` peut être implémentée comme ceci

```
(DEFUN MOVENTH* (n m l)
    (LET ((r (MOVENTH n m l)))
        (IF r r l)
    )
)
```

Voir aussi NTH, MOVENTH, démo `FileList.bbs`.

### 16.15.12 REMOVENTH

`REMOVENTH` supprime le n-ième élément d'une liste.

`(REMOVENTH` *`n list`*`)`

Renvoie une nouvelle liste où le *n*-ième élément de la liste (commençant par 0) a été supprimé.

Si *n* est NIL ou si le *n*-ième élément n'existe pas, retourne NIL.

Exemple : '`(REMOVENTH 1 (list 1 2 3))`' retourne ( 1 3 ).

Voir aussi NTH, REMOVENTH*, REPLACENTH, MOVENTH.

### 16.15.13 REMOVENTH*

`REMOVENTH*` est la version étoilée de `REMOVENTH`. La seule différence avec `REMOVENTH` est que `REMOVENTH*` retourne la liste initiale, si le *n*-ième élément n'existe pas.

`REMOVENTH*` peut être implémentée comme ceci

```
(DEFUN REMOVENTH* (n l)
    (LET ((r (REMOVENTH n l)))
        (IF r r l)
    )
)
```

Voir aussi NTH, REMOVENTH, démo `FileList.bbs`.

### 16.15.14 APPEND

`APPEND` concatène des listes.

`(APPEND [`*list ...*`])`

renvoie la concaténation de *list* ....

Exemple : '`(APPEND (list 1 2) (list 3 4) (list 5))`' donne ( 1 2 3 4 5 ).

Voir aussi LIST.

### 16.15.15 REVERSE

`REVERSE` renverse une liste.

`(REVERSE` *list*`)`

renvoie la liste renversée.

Exemple : '`(REVERSE (list 1 2 3))`' donne ( 3 2 1 ).

### 16.15.16 MAPFIRST

`MAPFIRST` applique une fonction à tous les éléments d'une liste.

`(MAPFIRST` *func list* `[...])`

Établit une liste dont les éléments sont le résultat de l'application d'une fonction appelant un par un en arguments les éléments d'une liste de départ. La longueur de la liste retournée est la longueur de la plus longue liste indiquée. Si une des listes indiquées est trop courte alors la liste est rallongée par des éléments NIL.

#### Exemples

| **Expression** | **Value** |
|---|---|
| `(MAPFIRST 1+ (LIST 1 2 3))` | `( 2 3 4 )` |
| `(MAPFIRST + (LIST 1 2 3) (LIST 2 3))` | `( 3 5 NIL )` |

### 16.15.17 SORTLIST

`SORTLIST` Trie les éléments d'une liste.

`(SORTLIST` *func list*`)`

Renvoie une copie de la liste indiquée qui a été triée en utilisant la fonction *func*. La fonction de tri doit prendre deux arguments, un pour chaque élément comparé, et renvoyer un entièr inférieur à zéro si le premier élément est plus petit que le second, un entier supérieur à zéro si le premier élément est plus grand que le second, ou 0 si les deux éléments sont égaux.

Exemple d'une fonction de tri :

```
(DEFUN cmp_str (x y)
    (COND
        ((< x y) -1)
        ((> x y) 1)
        (TRUE 0)
```

```
        )
    )
```

Maintenant vous pouvez trier une liste en appelant :

```
(SORTLIST cmp_str (LIST "salut" "excellent" "grand" "ok"))
```

qui retourne ( "excellent" "grand" "ok" "salut" ).

Voir aussi SORTLISTGT, MAPFIRST.

### 16.15.18 SORTLISTGT

`SORTLIST` tri les éléments de la liste

```
(SORTLISTGT gtfunc list)
```

Comme `SORTLIST` mais ici vous indiquez une fonction de tri qui renvoie une valeur différente de NIL si le premier élément est plus grand que le second, et NIL autrement.

Exemple : '`(SORTLISTGT > (LIST "salut" "excellent" "grand" "ok"))`' donne ( "excellent" "grand" "salut" "ok" ).

Voir aussi SORTLIST, MAPFIRST.

## 16.16 Fonctions de dialogue de saisie

Pour demander des informations à l'utilisateur, les fonctions suivantes peuvent être utilisées :

### 16.16.1 ASKFILE

`ASKFILE` demande à l'utilisateur de saisir un nom de fichier.

```
(ASKFILE title oktext default savemode)
```

Ouvre un sélecteur de fichier pour saisir le nom d'un fichier. Le titre de la fenêtre peut être placé dans *title*, le texte du bouton '`Ok`' dans *oktext*, et le nom du fichier initial dans *default*. Vous pouvez spécifier NIL pour l'un d'entre eux afin d'utiliser des valeurs par défauts. Le dernier argument *savemode* (booléen) permet d'ouvrir le sélecteur de fichier en mode sauvegarde. Ce mode devrait être utilisé lorsque le nom du fichier est destiné à y écrire quelque chose.

`ASKFILE` renvoie le nom de fichier écrit dans une chaîne ou NIL dans le cas où l'utilisateur aurait annulé l'opération.

Voir aussi ASKDIR, ASKSTR.

### 16.16.2 ASKDIR

`ASKDIR` demande à l'utilisateur de saisir le nom d'un répertoire.

```
(ASKDIR title oktext default savemode)
```

Ouvre le sélecteur de fichier pour saisir le nom d'un répertoire. Les arguments sont les même que ceux de `ASKFILE` (voir Section 16.16.1 [ASKFILE], page 127).

`ASKDIR` renvoie le nom du répertoire écrit dans une chaîne ou NIL dans le cas où l'utilisateur aurait annulé l'opération.

Voir aussi ASKFILE, ASKSTR.

### 16.16.3 ASKSTR

ASKSTR demande à l'utilisateur d'écrire un texte.

```
(ASKSTR title oktext default maxlen [secret])
```

Ouvre une fenêtre de saisie de texte. Le titre de la fenêtre, le texte du bouton 'Ok', et la valeur initiale peuvent être placés dans *title*, *oktext*, et *default* respectivement (chaîne ou NIL pour des valeurs par défaut), *maxlen* détermine le nombre maximum de caractères que l'utilisateur peut écrire. Si *secret* est spécifié et n'est pas NIL alors la chaîne saisie est rendue invisible en affichant un symbole indifférencié pour chaque caractère.

ASKSTR renvoie le texte entré ou NIL au cas où l'utilisateur annulerait l'opération.

Voir aussi ASKFILE, ASKDIR, ASKCHOICESTR, ASKINT.

### 16.16.4 ASKINT

ASKINT demande à l'utilisateur à saisir un nombre entier.

```
(ASKINT title oktext default min max)
```

Ouvre une fenêtre de saisie de nombre entier. Le titre de fenêtre et le texte du bouton 'Ok' peuvent être spécifiés dans *title* et *oktext* (chaîne ou NIL pour des valeurs par défaut). Dans *default* vous passez la valeur initiale du nombre entier ou NIL pour démarrer avec un champ d'édition vide. Dans *min* et *max* vous pouvez indiquer l'intervalle de validité du nombre entier. Les valeurs entrées en dehors de cet intervalle sont automatiquement rejetées. Utilisez NIL pour les valeurs par défaut de min et max.

ASKINT renvoie le nombre entier saisie ou NIL si l'utilisateur annule l'opération.

Voir aussi ASKSTR.

### 16.16.5 ASKCHOICE

ASKCHOICE demande à l'utilisateur à choisir un élément parmi d'autres.

```
(ASKCHOICE titre oktext choix defaut [titres])
```

Ouvre une fenêtre de saisie demandant à l'utilisateur de choisir un élément dans une liste. Vous pouvez placer le titre de la fenêtre et le texte du bouton 'Ok' dans *titre* et *oktext* (chaîne ou NIL pour des valeurs par défaut). Dans *choix* vous indiquez une liste de choix. Vous pouvez utiliser un format multi-colonnes pour présenter chaque choix comme une liste de sous-éléments. La valeur initiale peut être placée dans *defaut*, il s'agit de l'index de la ligne dans le mémo (en commençant par l'index 0 pour le premier élément). Utilisez NIL pour ne pas spécifier de valeur initiale. Si l'argument optionnel *titre* est spécifié et n'est pas NIL, alors un entête de colonne est affiché dans la liste en utilisant les titres contenus dans *titres*. Dans le cas de colonnes multiples, spécifiez *titles* comme une liste de titres de colonne.

*choix* et *titres* peuvent tous les deux être aussi représentés par un mémo et une chaîne (au lieu de listes). Dans ce cas ils sont convertis en listes par un appel automatique à `(MEMOTOLIST choices TRUE)` (voir Section 16.13.3 [MEMOTOLIST], page 119) et `(STRTOLIST titles)` (voir Section 16.12.23 [STRTOLIST], page 113) respectivement.

ASKCHOICE renvoie l'index de l'élément choisi ou NIL si l'utilisateur a annulé l'opération.

### Exemple

```
(LET ((items (LIST "Première entrée" 2 3.14 "Dernière entrée")) index)
    (SETQ index (ASKCHOICE "Choisissez un élément" "Ok" items NIL))
    (IF index
        (PRINTF "L'utilisateur a choisi l'élément numéro %i avec le contenu <%s>\n"
            index (STR (NTH index items))
        )
    )
)
```

Considérez le cas où vous voulez demander à l'utilisateur de sélectionner un enregistrement particulier dans une table. La table doit se nommer '`Article`' et posséder les champs '`Nom`', '`Quantite`', et '`Prix`'. Le bout de code suivant montre comment utiliser ASKCHOICE pour sélectionner un enregistrement avec un prix inférieur à 10 et ordonnés par leur Nom :

```
(LET ((query (SELECT Article, Nom, Quantite, Prix from Article
      WHERE (> Prix 10) ORDER BY Nom))
     (recs (MAPFIRST FIRST (REST query))) ; record pointers
     (items (MAPFIRST REST (REST query))) ; choices
     (titles (REST (FIRST query))) ; titles
     (index (ASKCHOICE "Sélection" "Ok" items NIL titles))
     (rec (NTH index recs)))
     ; now rec holds the selected record (or NIL on cancel)
)
```

Voir aussi ASKCHOICESTR, ASKOPTIONS.

## 16.16.6 ASKCHOICESTR

`ASKCHOICESTR` demande à l'utilisateur de choisir un chaîne parmi une liste prédéfinie.

`(ASKCHOICESTR` *titre oktext chaines defaut* [*titres*]`)`

Ouvre une fenêtre de saisie demandant à l'utilisateur de choisir une chaîne dans une liste ou d'en saisir une dans un champ texte séparé. Vous pouvez placer le titre de fenêtre et le texte du bouton '`Ok`' dans *titre* et *oktext* (chaîne ou NIL pour des valeurs par défaut). Dans *chaines* vous indiquez la liste de choix. Vous pouvez utiliser un format multi-colonnes pour présenter chaque choix comme une liste de sous-éléments. La valeur initiale du champ texte peut être placée dans *defaut* (chaîne ou NIL pour un champ texte vide). Si l'argument optionnel *titre* est spécifié et n'est pas NIL, alors un entête de colonne est affiché dans la liste en utilisant les titres contenus dans *titres*. Dans le cas de colonnes multiples, spécifiez *titles* comme une liste de titres de colonne.

*choix* et *titres* peuvent tous les deux être aussi représentés par un mémo et une chaîne (au lieu de listes). Dans ce cas ils sont convertis en listes par un appel automatique à `(MEMOTOLIST` *choices* `TRUE)` (voir Section 16.13.3 [MEMOTOLIST], page 119) et `(STRTOLIST` *titles*`)` (voir Section 16.12.23 [STRTOLIST], page 113) respectivement.

`ASKCHOICESTR` renvoie la chaîne choisie ou NIL si l'utilisateur a fermé la fenêtre.

### Exemple

```
(LET ((strings (LIST "Claudia" "Mats" "Ralphie")) likebest)
```

```
        (SETQ likebest
            (ASKCHOICESTR "Qui aimes-tu le plus ?" "Ok" strings
        "Mes chiens!"
    )
        )
        (IF likebest (PRINTF "L'utilisateur a choisi <%s>\n" likebest))
    )
```

Voir aussi ASKCHOICE, ASKOPTIONS.

### 16.16.7 ASKOPTIONS

`ASKOPTIONS` demande à l'utilisateur de sélectionner plusieurs éléments dans une liste.

`(ASKOPTIONS` *titre oktext options selectionnes* [*titres*]`)`

Ouvre une fenêtre de saisie demandant à l'utilisateur de sélectionner une ou plusieurs options dans une liste. Vous pouvez placer le titre de la fenêtre et le texte du bouton '`Ok`' dans *titre* et *oktext* (chaîne ou NIL pour des valeurs par défaut). Dans *options* vous indiquez la liste d'options. Vous pouvez utiliser un format multi-colonnes pour présenter chaque option comme une liste de sous-éléments. La sélection initiale peut être spécifiée dans *selectionnes* sous forme d'une liste de nombres entiers spécifiant l'index des lignes correspondantes dans *options* qui doivent être sélectionnées au début. Utilisez NIL pour tous les éléments ne devant pas être présélectionnés. Si l'argument optionnel *titre* est spécifié et n'est pas NIL, alors un entête de colonne est affiché dans la liste en utilisant les titres contenus dans *titres*. Dans le cas de colonnes multiples, spécifiez *titles* comme une liste de titres de colonne.

*choix* et *titres* peuvent tous les deux être aussi représentés par un mémo et une chaîne (au lieu de listes). Dans ce cas ils sont convertis en listes par un appel automatique à `(MEMOTOLIST` *choices* `TRUE)` (voir Section 16.13.3 [MEMOTOLIST], page 119) et `(STRTOLIST` *titles*`)` (voir Section 16.12.23 [STRTOLIST], page 113) respectivement.

`ASKOPTIONS` renvoie une liste de nombres entiers correspondant à l'index des éléments sélectionnés ou NIL dans le cas où l'utilisateur aurait annulé l'opération ou n'aurait pas choisi d'élément.

#### Exemple

```
(LET ((options (LIST "Salva Mea" "Insomnia" "Don't leave" "7 days & 1 week"))
        (selected (LIST 0 1 3))
    )
    (SETQ selected (ASKOPTIONS "Choisir un titre" "Ok" options selected))
    (IF selected
        (
            (PRINTF "L'utilisateur a choisi les titres suivants :\n")
            (DOLIST (i selected)
                (PRINTF "\tnum: %i contenu : <%s>\n" i (STR (NTH i options)))
            )
        )
    )
)
```

### 16.16.8 ASKBUTTON

`ASKBUTTON` demande à l'utilisateur de cliquer sur un bouton.

`(ASKBUTTON` *`title text buttons canceltext`*`)`

Ouvre une fenêtre de dialogue avec un titre (chaîne ou NIL pour un titre par défaut) et une description (chaîne ou NIL s'il n'y en a pas). La fonction attend jusqu'à ce que l'utilisateur clique sur un des boutons indiqués dans *buttons* (liste de chaînes) ou le bouton '`Annuler`'. Le texte du bouton d'annulation peut être spécifié dans *canceltext*. Si vous indiquez NIL un texte par défaut basé sur le nombre de boutons que vous avez indiqué est employé.

`ASKBUTTON` renvoie le numéro du bouton cliqué (à partir de 0, pour le bouton à l'extrême gauche) ou NIL si l'utilisateur clique sur le bouton '`Annuler`'.

#### Exemples

```
(LET ((buttons (LIST "A la maison" "Au lit" "Devant mon ordinateur")) index)
    (SETQ index (ASKBUTTON "Veuillez répondre :"
        "Où voudriez-vous être demain ?" buttons "Je ne sais pas")
    )
    (IF index
        (PRINTF "Choix de l'utilisateur : <%s>\n" (NTH index buttons))
    )
)

(ASKBUTTON "Info" "BeeBase est génial !" NIL NIL)
```

Voir aussi ASKCHOICE.

### 16.16.9 ASKMULTI

`ASKMULTI` demande à l'utilisateur de saisir divers types d'informations.

`(ASKMULTI` *`title oktext itemlist`*`)`

`ASKMULTI` est une fonction de dialogue à usages multiples. Elle ouvre une fenêtre avec le titre indiqué, un ensemble d'objets d'IHM pour l'édition de données, et deux boutons ('`Ok`' et '`Annuler`') pour fermer la requête. Le texte du bouton '`Ok`' peut être placé dans *oktext* (chaîne ou NIL pour le texte de défaut). L'ensemble des objets d'IHM sont spécifiés dans *itemlist* qui est une liste où chaque élément a l'une des formes suivantes :

```
(LIST title "String" initial [help [secret]])   pour une ligne de texte,
(LIST title "Memo" initial [help])     pour plusieurs lignes de texte,
(LIST title "Integer" initial [help])  pour un nombre entier,
(LIST title "Real" initial [help])     pour un réel,
(LIST title "Date" initial [help])     pour une date,
(LIST title "Time" initial [help])     pour une heure,
(LIST title "Bool" initial [help])     pour un booléen,
(LIST title "Choice" initial
    (LIST choice ...) [help]
)                                      pour un champ choix.
(LIST title "ChoiceList" initial
    (LIST choice ...) [help]
```

```
)                                       pour choisir un élément dans une liste.
(LIST title "Options" initial
    (LIST option ...) [help]
)                                       pour choisir plusieurs éléments dans une liste.
non-list-expr                           pour un texte non modifiable
```

Le titre (chaîne ou NIL s'il n'y a pas de titre) sera affiché à la gauche de l'objet d'IHM. Si la valeur initiale est NIL la valeur par défaut est utilisée (par exemple un champ Texte vide). Pour les champs de choix la valeur initiale doit être l'index (commençant à 0) de l'entrée initialement active. Pour les champs de liste de choix la valeur initiale peut être NUL (pas d'élément actif). Pour les champs d'option la valeur initiale doit être une liste d'entiers représentants les indices (commençant à 0) des éléments sélectionnés au départ. Le champ d'aide optionnel peut être utilisé pour renseigner l'utilisateur sur l'utilisation du champ. Pour les champs text un paramètre supplémentaire '`secret`' peut être spécifié. S'il n'est pas NIL alors le contenu du champ texte est masqué par des symboles indifférents pour chacun de ses caractères.

`ASKMULTI` renvoie une liste de valeurs que l'utilisateur a édités et validés en cliquant sur le bouton '`Ok`'. Chaque valeur d'un champ a le même format que celui de la valeur initiale. Par exemple pour le choix d'une liste de champs la valeur résultat est l'index de l'élément choisi (ou NIL si aucun élément n'a été choisi) ou pour une option, la valeur résultat est une liste de nombres entiers représentant les index des éléments choisis. Pour le ttexte non modifiable une valeur NIL est retournée.

Par exemple si vous avez indiqué un champ date, un champ texte non modifiable, un champ choix, un champ option et une champ texte avec la valeur initiale "world", et que l'utilisateur a écrit 11.11.1999, choisi l'entrée avec l'index numéro 2, choisi le 3ème et 4ème élément dans le champ option, et laissé le champ texte intact alors la fonction renvoie la liste ( 11.11.1999 NIL 2 ( 3 4 ) "world" ).

Si l'utilisateur a annulé l'opération, NIL est retourné.

### Exemple

```
(ASKMULTI "Veuillez répondre :" NIL (LIST
    (LIST "N_om" "String" "")
    (LIST "_Naissance" "Date" NIL)
    (LIST "_Sexe" "Choice" 0 (LIST "homme" "femme"))
    (LIST "Possède une _voiture ?" "Bool" NIL)
    (LIST "_Aime" "Options" (LIST 0 2)
        (LIST "Bière" "Vin" "Whisky" "Wodka" "Schnaps")
    ))
)
```

Consultez le projet '`AskDemo.bbs`' pour d'autres exemples.

## 16.17 Fonctions d'E/S

Cette section liste les fonctions et les variables d'entrées/sortie des données (par exemple l'impression).

### 16.17.1 FOPEN

`FOPEN` ouvre un fichier en lecture/écriture.

`(FOPEN` *nomfichier mode* `[`*encodage*`])`

Ouvre le fichier spécifié par *nomfichier* (string). Le paramètre *mode* (string) contrôle le mode d'accès. Utilisez '`"w"`' pour ouvrir un fichier en écriture, '`"a"`' pour y ajouter des données ou '`"r"`' pour y lire des données. Vous pouvez aussi utiliser d'autres drapeaux (ou combinaison de drapeaux) comme '`"r+"`' pour lire et écrire. Aucun contrôle de validité des drapeaux n'est effectué. Si le fichier ne peut être ouvert, l'instruction renvoie NIL.

Le paramètre optionnel *encodage* contrôle l'encodage du fichier et peut prendre l'une des chaînes suivantes pour valeur :

'`"none"`' : Aucune interprétation de caractère n'est effectuée. Utilisez ceci pour les fichiers binaires.

'`"UTF-8"`' :
Le texte est encodé en UTF-8. La lecture et l'écriture convertissent depuis/en UTF-8.

'`"locale"`' :
Le texte est encodé selon les réglages de votre système. Sur Linux, c'est le contenu des variables d'environnement `LANG` et `LC_*`, voir `man locale`. Sur Windows, c'est la page de code du système. Sur Amiga, c'est l'encodage à 8-bit par défaut.

'`"8-bit"`' :
Le texte est encodé selon l'encodage 8-bit par défaut. Sur Linux et Windows, il s'agit de l'encodage ISO-8859-1 (latin 1). Sur Amiga, il s'agit de l'encodage 8-bit par défaut du système (comme '`locale`').

'`"auto"`' : L'encodage est détecté automatiquement. Si le fichier est lisible alors l'encodage est déterminé comme suit : Si l'intégralité du contenu est conforme à UTF-8 alors '`"UTF-8"`' est utilisé. Sinon, si la locale du système n'est pas UTF-8 alors '`"locale"`' est utilisé. Autrement, c'est '`"8-bit"`' qui est utilisé. Lors de l'écriture, si l'encodage n'a pas encore été déterminé, alors la première locale du système rencontrée est essayée. Si aucune erreur de conversion ne se produit, c'est '`"locale"`' qui est utilisé, sinon '`"UTF-8"`'.

Si aucun paramètre *encodage* n'est spécifié, c'est '`"auto"`' qui est utilisé.

En cas de succès, `FOPEN` renvoie un descripteur de fichier. En cas d'échec, c'est NIL qui est renvoyé. Si *nomfichier*, *mode* ou *encodage* sont à NIL, c'est NIL qui est renvoyé.

#### Exemples

'`(FOPEN "index.html" "w" "utf-8")`' ouvre et renvoie un descripteur de fichier pour écrire dans le fichier '`index.html`' avcec l'encodage UTF-8.

'`(FOPEN "output.txt" "a+")`' ouvre le fichier '`output.txt`' pour y ajouter des données dans le même encodage que celui du fichier. Remarquez que si vous spécifiez uniquement '`"a"`' pour mode, BeeBase peut ne pas être en mesure de lire le fichier et de déterminer son encodage. Dans ce cas, l'encodage sera déterminé au moment de l'écriture (et pourrait être différent de celui de la première partie du fichier).

Voir aussi FCLOSE, stdout, FFLUSH.

### 16.17.2 FCLOSE

`FCLOSE` ferme un fichier.

`(FCLOSE` *file*`)`

Ferme le fichier donné. Renvoie 0 en cas de succès, NIL si une erreur s'est produite. Si *file* est NIL alors 0 est retourné (aucune erreur). Après la fermeture du fichier l'accès au descripteur de fichier est une opération illégale et interrompt l'exécution du programme avec un message d'erreur.

Voir aussi FOPEN, FFLUSH.

### 16.17.3 stdout

La variable globale `stdout` conserve le descripteur de fichier vers la sortie standard de BeeBase. Le nom du fichier de sortie peut être réglé depuis le menu '`Programme - fichier de sortie`' (voir Section 7.2.13 [Program output file], page 39).

Le fichier de sortie est ouvert au premier accès à cette variable (par exemple en appelant '`(FPRINTF stdout ...)`' ou '`(PRINTF ...)`'). Le fichier n'est ouvert automatiquement au démarrage de BeeBase, afin d'éviter une ouverture si aucune sortie n'est générée. Par exemple lorsque vous voulez seulement effectuer quelques calculs et changer le contenu de certains enregistrements.

Lors de l'ouverture du fichier de sortie, le paramètre de mode peut être soit '`"w"`', soit '`"a+"`' en fonction du réglage '`Ajouter`' du menu '`Programme - Fichier de sortie`'. L'encodage est réglé à '`"auto"`'.

Si BeeBase ne peut pas ouvrir le fichier de sortie le programme s'arrête et un message d'erreur est produit.

Voir aussi FOPEN, PRINTF.

### 16.17.4 PRINT

`PRINT` convertit une expression en chaîne et l'affiche.

`(PRINT` *elem*`)`

Convertit la valeur de *elem* en chaîne lisible et l'imprime dans `stdout`. Cette fonction existe principalement pour le débogage (la mise au point). This function is not considered having a side effect such that it can be used e.g. in debugging a comparison function.

Voir aussi PRINTF, stdout, Fonction de comparaison.

### 16.17.5 PRINTF

`PRINTF` imprime une chaîne formater.

`(PRINTF` *format* `[`*expr ...*`])`

Formate une chaîne en utilisant les données du format chaîne et arguments pour l'imprimer dans `stdout`. Le formatage est fait comme dans `SPRINTF` (voir Section 16.12.34 [SPRINTF], page 116).

`PRINTF` renvoie le nombalors NIL est retourné.

Cette fonction n'est pas considérée comme ayant un effet secondaire pour qu'elle puissepeut être utilisée par ex. dans le débogage d'une fonction de comparaison.

Exemple : '`(PRINTF "%i jours et %i semaine" 7 1)`' imprime la chaîne "7 jours et 1 semaine" dans `stdout` et renvoie 17.

Voir aussi PRINT, FPRINTF, stdout, Fonction de comparaison.

### 16.17.6 FPRINTF

`FPRINTF` imprime une chaîne formatée dans un fichier.

```
(FPRINTF file format [expr ...])
```

Formate une chaîne en utilisant les données du format chaîne et arguments pour l'imprimer dans le fichier spécifié. Le formatage est fait comme dans `SPRINTF` (voir Section 16.12.34 [SPRINTF], page 116).

`FPRINTF` renvoie le nombre de caractères en sortie ou NIL en cas d'échec. Si *file* est NIL alors `FPRINTF` renvoie toujours le nombre de caractères potentiellement écrits mais rien n'est réellement écrit en sortie. Si *format* est NIL alors NIL est retourné.

Voir aussi PRINTF, FOPEN.

### 16.17.7 FERROR

`FERROR` vérifie si un fichier produit une erreur d'entrée/sortie.

```
(FERROR file)
```

renvoie TRUE si une erreur s'est produite, NIL autrement. Si '`file`' est NIL, NIL est retourné.

Voir aussi FEOF, FOPEN, FCLOSE.

### 16.17.8 FEOF

`FEOF` vérifie l'état de fin de fichier.

```
(FEOF file)
```

Examine l'indicateur de fin de fichier d'un fichier donné et renvoie TRUE s'il est vrai. Autrement NIL est retourné. Si le '`file`' est NIL, NIL est retourné.

Voir aussi FERROR, FTELL, FOPEN, FCLOSE.

### 16.17.9 FSEEK

`FSEEK` définit la position de lecture/écriture d'un fichier.

```
(FSEEK file offset whence)
```

Définit la position de lecture/écriture pour un fichier donné. La nouvelle position, mesurée en octets, est obtenue en ajoutant des *offset* octets à la position spécifiée par *whence*. Si *whence* est placé sur SEEK_SET, SEEK_CUR ou SEEK_END, *offset* se reporte respectivement au début du fichier, à la position actuelle ou à la fin du fichier.

En cas de succès, `FSEEK` renvoie 0. Sinon NIL est retourné et la position dans le fichier reste inchangée. Si *file*, *offset* ou *whence* sont NIL ou si *whence* n'est pas une des constantes SEEK_SET, SEEK_CUR ou SEEK_END, alors NIL est retourné.

Veuillez noter qu'après une lecture l'appel à `FSEEK` avec une valeur *whence* pour SEEK_BUR n'est supporté que pour l'encodage '`"none"`'.

Voir aussi FTELL, FOPEN, Constantes prédéfinies.

### 16.17.10 FTELL

`FTELL` renvoie la position de lecture/écriture d'un fichier.

```
(FTELL file)
```

Détermine la position de lecture/écriture courante d'un fichier donné par rapport au début du fichier et la renvoie comme un nombre entier. Si une erreur se produit ou si '`file`' est NIL, alors NIL est retourné.

Veuillez noter qu'après une lecture l'appel à `FSEEK` avec une valeur *whence* pour SEEK_BUR n'est supporté que pour l'encodage '`"none"`'.

Voir aussi FSEEK, FOPEN, FEOF.

### 16.17.11 FGETCHAR

`FGETCHAR` lit un caractère à partir d'un fichier.

```
(FGETCHAR file)
```

Renvoie le prochain caractère à partir d'un fichier donné de la chaîne ou NIL si *file* est NIL, si la fin du fichier à été atteinte ou si une erreur s'est produite. Si le prochain caractère est un caractère nul, une chaîne vide est retournée.

Voir aussi FGETCHARS, FGETSTR, FPUTCHAR.

### 16.17.12 FGETCHARS

`FGETCHARS` lit des caractères à partir d'un fichier.

```
(FGETCHARS num file)
```

renvoie une chaîne contenant les *num* prochains caractères depuis un fichier donné Si la fin du fichier a été atteinte avant la lecture des *num* caractères ou si un caractère nul a été lu alors les caractères déjà lus sont retournés. Si *num* ou *file* sont NIL, si *num* est négative, si la fin du fichier a été atteinte avant de lire le premier caractère ou si une erreur de lecture c'est produite, NIL est retourné.

Voir aussi FGETCHAR, FGETSTR.

### 16.17.13 FGETSTR

`FGETSTR` lit une chaîne à partir d'un fichier.

```
(FGETSTR file)
```

renvoie la prochaine ligne d'un fichier donné comme une chaîne ou NIL si *file* est NIL, la fin du fichier a été atteint, ou une erreur s'est produite. La fin d'une ligne est détectée soit par une nouvelle ligne de caractères, soit par la lecture d'un caractère nul, soit si la fin du fichier est détectée. Dans l'un ou l'autre cas la chaîne ne contient aucune nouvelle ligne de caractères.

Voir aussi FGETCHAR, FGETCHARS, FGETMEMO, FPUTSTR.

### 16.17.14 FGETMEMO

`FGETMEMO` lit un mémo à partir d'un fichier.

```
(FGETMEMO file)
```

renvoie un mémo qui contient le contenu d'un fichier donné jusqu'au prochain caractère nul ou jusqu'à la fin du fichier. Si *file* est NIL, la fin du fichier a été atteinte avant de lire tous les caractères, ou une erreur s'est produite alors NIL est retourné.

Voir aussi FGETSTR, FPUTMEMO.

### 16.17.15 FPUTCHAR

`FPUTCHAR` écrit un caractère dans un fichier.

```
(FPUTCHAR str file)
```

Écrit le premier caractère de *str* dans un fichier donné. Si *str* est vide, un caractère nul est écrit, si *str* ou *file* sont NIL, rien ne se produit. Renvoie *str* ou NIL au cas où une erreur se produirait.

Voir aussi FPUTSTR, FGETCHAR.

### 16.17.16 FPUTSTR

`FPUTSTR` écrit une chaîne dans un fichier.

```
(FPUTSTR str file)
```

Imprime *str* ainsi qu'une nouvelle ligne de caractères dans un fichier donné. Si *str* ou *file* sont NIL, rien ne se produit. Renvoie *str* ou NIL au cas où une erreur se produirait.

Voir aussi FPUTCHAR, FPUTMEMO, FGETSTR.

### 16.17.17 FPUTMEMO

`FPUTMEMO` écrit un mémo dans un fichier.

```
(FPUTMEMO memo file)
```

Imprime *memo* dans un fichier donné. Si *memo* ou *file* sont NIL, rien ne se produit. Renvoie *memo* ou NIL au cas où une erreur de sortie se produirait.

Voir aussi FPUTSTR, FGETMEMO.

### 16.17.18 FFLUSH

`FFLUSH` purge les données en attente vers un fichier.

```
(FFLUSH file)
```

Purge toutes les données en attente d'écriture pour le fichier spécifié. Renvoie 0 en cas de succès, NIL si une erreur se produit. Si *file* est NIL alors 0 est retourné (aucune erreur).

Voir aussi FOPEN, FCLOSE.

## 16.18 Fonctions sur les enregistrements

Cette section énumère les fonctions qui traitent des enregistrements.

### 16.18.1 NEW

`NEW` alloue un nouvel enregistrement pour une table.

```
(NEW table init)
```

Alloue un nouvel enregistrement pour une table donnée. Le paramètre *init* spécifie l'enregistrement qui doit être utilisé pour initialiser le nouvel enregistrement. La valeur NIL représente l'enregistrement initial.

`NEW` renvoie un indicateur d'enregistrement pour le nouvel enregistrement.

La fonction `NEW` a un effet secondaire qui initialise le programme pointant les enregistrements d'une table donnée (voir Section 5.2 [Tables], page 20) vers un nouvel enregistrement.

Exemple : '`(NEW table NIL)`' alloue un nouvel enregistrement dans une table donnée et l'initialise avec l'enregistrement initial.

Voir aussi NEW*, DELETE, Tables.

### 16.18.2 NEW*

`NEW*` est la version étoilée de `NEW` (voir Section 16.18.1 [NEW], page 137).

`(NEW* table init)`

`NEW*` vérifie si vous avez spécifié un déclencheur de '`Création`' pour la table donnée (voir Section 16.29.8 [New trigger], page 160). Si c'est le cas, cette fonction est appelée pour allouer l'enregistrement et son résultat est retourné. Le paramètre *init* peut être utilisé pour spécifier l'enregistrement avec lequel le nouvel enregistrement devrait être initialisé (utilisez NIL pour un enregistrement initial).

Si aucun déclencheur n'a été spécifié, la fonction se comporte comme la fonction `NEW`.

Avertissement : Avec cette fonction il est possible d'écrire des boucles sans fin, par exemple si vous avez défini un déclencheur de '`Création`' pour une table et qu'il appelle `NEW*` pour allouer les enregistrements.

Voir aussi NEW, DELETE*.

### 16.18.3 DELETE

`DELETE` supprime les enregistrements dans une table.

`(DELETE table confirm)`

Supprime les enregistrements du programme en cours d'une table donnée après une demande de confirmation facultative. Le premier argument indique la table dans laquelle les enregistrements du programme en cours devraient être supprimés, le deuxième argument est une expression booléenne. Si c'est NIL alors l'enregistrement est supprimé sans confirmation, sinon l'état du menu préférences '`Confirmer la suppression des enregistrements`' est vérifié. S'il n'est pas coché, l'enregistrement est supprimé sans confirmation, autrement une fenêtre de confirmation standard apparaît vous demandant si vous voulez vraiment effacer cet enregistrement. Si l'utilisateur clique sur le bouton annuler, l'enregistrement ne sera pas supprimé.

Le code renvoyé par la fonction `DELETE` reflète l'action choisie. S'il renvoie TRUE alors l'enregistrement a été supprimé, sinon (l'utilisateur a annulé l'opération) NIL est retourné.

À propos de la suppression, `DELETE` place le pointeur du programme d'enregistrement (voir les voir Section 5.2 [Tables], page 20) de la table spécifiée sur NIL.

Exemple : '`(DELETE table NIL)`' supprime, sans confirmation, l'enregistrement dans la table de données courante.

Voir aussi DELETE*, DELETEALL, NEW, Tables.

### 16.18.4 DELETE*

`DELETE*` est la version étoilée de `DELETE` (voir Section 16.18.3 [DELETE], page 138).

```
(DELETE* table confirm)
```

`DELETE*` vérifie si vous avez spécifié un déclencheur de '`Suppression`' pour la table donnée (voir Section 16.29.9 [Delete trigger], page 160). Si c'est le cas, alors cette fonction est appelée pour supprimer l'enregistrement et son résultat est retourné. Le paramètre *confirm* peut être utilisé pour indiquer si le déclencheur appelle la fenêtre de confirmation avant de supprimer l'enregistrement.

Si aucun déclencheur n'a été spécifié, la fonction se comporte comme la fonction `DELETE`.

Avertissement : Avec cette fonction il est possible d'écrire des boucles sans fin, par exemple si vous avez défini un déclencheur de '`Suppression`' pour une table et qu'il appelle `DELETE*` pour supprimer l'enregistrement.

Voir aussi DELETE, DELETEALL, NEW*.

### 16.18.5 DELETEALL

`DELETEALL` supprime tous les enregistrements d'une table.

```
(DELETEALL table[*])
```

Supprime tous les enregistrements de la table indiquée. si vous ajoutez une étoile derrière le nom de la table seul les enregistrements du filtre de la table en cours seront supprimés. Il n'y a pas de fenêtre de confirmation avant de supprimer les enregistrements.

`DELETEALL` renvoie TRUE en cas de succès pour la suppression de tous les enregistrements, sinon NIL est retourné. Si la *table* est NIL alors NIL est retourné.

Exemple : '`(DELETEALL table*)`' supprime, en utilisant le filtre de la table, tous les enregistrements de la table de données.

Voir aussi DELETE, Tables.

### 16.18.6 GETMATCHFILTER

`GETMATCHFILTER` renvoie l'état du filtre d'enregistrement.

```
(GETMATCHFILTER rec)
```

Renvoie TRUE si les enregistrements disposent d'un filtre dans la table, NIL dans les autres cas. Si le filtre de la table courante n'est pas actif, alors TRUE est retourné. Si *rec* est NIL (l'enregistrement initial) alors NIL est retourné.

Voir aussi SETMATCHFILTER, GETISSORTED, GETFILTERSTR, SETFILTERSTR.

### 16.18.7 SETMATCHFILTER

`SETMATCHFILTER` définit l'état du filtre d'un enregistrement.

```
(SETMATCHFILTER rec on)
```

Change l'état du filtre de l'enregistrement indiqué à la valeur *on*. `SETMATCHFILTER` renvoie le nouvel état du filtre de l'enregistrement donné. Le nouvel état peut être différent de ce qui était prévu parce que le réglage de l'état du filtre vers NIL fonctionne seulement quand le filtre de la table correspondante est en activité, sinon TRUE est retourné. Appelez

`SETMATCHFILTER` avec la valeur NIL pour *rec* (l'enregistrement initial) renverra toujours NIL.

Voir aussi GETMATCHFILTER, SETISSORTED, GETFILTERSTR, SETFILTERSTR.

### 16.18.8 GETISSORTED

`GETISSORTED` renvoie l'état du type d'enregistrement.

```
(GETISSORTED rec)
```

Renvoie TRUE si l'enregistrement spécifie le type de tri qui a été défini pour sa table, NIL dans les autres cas. Si *rec* est NIL alors NIL est retourné.

Voir aussi SETISSORTED, GETMATCHFILTER, REORDER, GETORDERSTR, SETORDERSTR, Fonction de comparaison.

### 16.18.9 SETISSORTED

`SETISSORTED` modifie l'indicateur de tri de l'enregistrement.

```
(SETISSORTED rec on)
```

Change l'indicateur de tri de l'enregistrement spécifié par la valeur *on*. Utilisez cette fonction si vous pensez que certains enregistrements sont dans le bon ordre (*on* = TRUE) ou s'ils doivent être réordonnés (*on* = NIL). Le réordonnancement de tous les enregistrements qui ne sont pas triés peut être réalisé en appelant la fonction `REORDER` (voir voir Section 16.20.4 [REORDER], page 144).

`SETISSORTED` renvoie la nouvelle valeur de l'indicateur pour l'enregistrement donné. Appeler `SETISSORTED` avec la valeur NIL pour *rec* (l'enregistrement initial) renverra NIL.

Pour un exemple sur la façon d'employer cette fonction, voir Section 16.29.10 [Comparison function], page 161.

Voir aussi GETISSORTED, SETMATCHFILTER, REORDER, GETORDERSTR, SETORDERSTR, Fonction de comparaison.

### 16.18.10 GETREC

`GETREC` renvoie la référence d'enregistrement d'une expression.

```
(GETREC expr)
```

Renvoie l'enregistrement qui a été utilisé lors de la création de *expr*. Les expressions créées à partir d'un champ dans une table ont l'enregistrement d'où elles proviennent défini comme référence d'enregistrement.

Si *expr* est une liste et n'a elle-même pas de référence d'enregistrement alors `GETREC` renvoie l'enregistrement du premier élément de la liste qui a une référence d'enregistrement. `GETREC` n'examine cependant pas les sous-listes.

Si aucune référence d'enregistrement n'a été trouvée, `GETREC` renvoie NIL.

La fonctionnalité d'examen des éléments d'une liste est pratique lors de l'obtention de la référence d'enregistrement d'une ligne dans une requête select-from-where. Par exemple, le fragment de code suivant renvoie l'enregistrement qui a été utilisé lors de la création de la 7ème ligne dans une requête select-from-where :

```
(GETREC (NTH 7 (SELECT field FROM table)))
```

Cette fonction est utile, par ex. dans une fonction de déclenchement de suppression de tri pour une liste virtuelle, voir Section 16.29.17 [Sort drop trigger], page 165.

Voir aussi SETREC, RECORD, Déclencheur de tri-déplacement.

### 16.18.11 SETREC

`SETREC` définit la référence d'enregistrement d'une expression.

```
(SETREC expr record)
```

Renvoie *expr* avec une référence d'enregistrement définie sur *record*.

`SETREC` est utile lorsqu'une expression doit être associée à un certain enregistrement, par ex. lors du calcul d'informations dérivées d'un champ dans une table. Par exemple, dans une table '`Person`' avec un champ booléen '`Female`', une requête select-from-where pourrait mapper '`Female`' aux chaînes `"F"` et `"M"` :

```
SELECT (SETREC (IF Female "F" "M") Person) FROM Person
```

Avec la commande `SETREC`, `"F"` et `"M"` sont associés à l'enregistrement dont ils sont dérivés, ce qui, par exemple, permet à une liste virtuelle utilisant ce support de requête d'ouvrir l'enregistrement correspondant par un double clic (voir Section 15.3.3 [Field object editor], page 73)

Voir aussi GETREC.

### 16.18.12 RECNUM

`RECNUM` renvoie le numéro d'un enregistrement.

```
(RECNUM record)
```

Renvoie le numéro d'un enregistrement donné. Veuillez noter que la numérotation des enregistrements est différente de celle utilisée par exemple pour les listes. Pour les listes, les chaînes et autres, le compte commence à zéro. Toutefois, pour les enregistrements, il commence à 1 pour le premier enregistrement. Le numéro 0 est réservé à l'enregistrement initial. Cela semble être contradictoire avec le reste des fonctions de programmation de BeeBase, mais prend ici vraiment son sens lorsque les numéros d'enregistrement sont également utilisés dans la fenêtre d'affichage.

Voir aussi RECORDS, MOVEREC, INT.

### 16.18.13 MOVEREC

`MOVEREC` déplace un enregistrement vers une nouvelle position.

```
(MOVEREC record pos)
```

Déplace l'enregistrement donné à la position donnée de la table de sorte que le numéro d'enregistrement devienne *pos*. Notez que les numéros d'enregistrement commencent par 1 pour la première position. Renvoie le nouveau numéro d'enregistrement ou NIL si le déplacement a échoué, par ex. si *pos* n'est pas compris entre 1 et le nombre d'enregistrements dans la table.

Si l'enregistrement a été déplacé, efface également l'état trié de l'enregistrement. Notez que si la table a un ordre défini, la réorganisation des enregistrements dans la table remettra l'enregistrement à sa place. Ainsi, cette fonction est principalement utile pour les tables qui n'ont pas d'ordre défini.

Cette fonction est utile dans une fonction de déclenchement de suppression de tri pour une liste virtuelle, voir Section 16.29.17 [Sort drop trigger], page 165.

Voir aussi RECNUM, GETISSORTED, Déclencheur de tri-déplacement.

### 16.18.14 COPYREC

`COPYREC` copie les enregistrements.

```
(COPYREC rec source)
```

Copie le contenu de l'enregistrement *source* dans l'enregistrement *rec*. Si *source* est NIL alors *rec* est initialisé à la valeur de l'enregistrement initial. Si *rec* est NIL alors, un message d'erreur est généré.

`COPYREC` renvoie *rec*.

Voir aussi NEW.

## 16.19 Fonctions sur les champs

Cette section énumère les fonctions qui travaillent sur les champs d'une table.

### 16.19.1 FIELDNAME

`FIELDNAME` renvoie le nom d'un champ.

```
(FIELDNAME field)
```

Renvoie une chaîne contenant le nom d'un champ indiqué.

Voir aussi TABLENAME

### 16.19.2 MAXLEN

`MAXLEN` renvoie la taille maximum du champ d'une chaîne.

```
(MAXLEN string-field)
```

Renvoie le nombre maximum de caractères que le champ d'une chaîne donné peut contenir.

Voir aussi LEN.

### 16.19.3 GETLABELS

`GETLABELS` renvoie toutes les entrées d'un choix ou d'une chaîne.

```
(GETLABELS field)
```

Renvoie les entrées d'un champ choix ou chaîne donné. Dans le cas d'un champ choix les entrées que vous avez entrées saisies la fenêtre de saisie du champ (voir Section 15.2.2 [Type specific settings], page 67) sont retournées, dans le cas du champ chaîne les entrées statiques saisies dans la liste déroulante (voir Section 15.3.3 [Field object editor], page 73) sont retournées (notez que cette fonction n'est utile que pour des entrées statiques).

Les entrées sont retournées dans une seule chaîne et sont séparées par un caractère de retour à la ligne.

Par exemple, si vous considérez que vous avez un champ choix avec les entrées '`Voiture`', '`Maison`', et '`Huile`'. En appelant `GETLABELS` sur ce champ vous obtiendrez le résultat `"Voiture\nMaison\nHuile"` dans une chaîne.

Note: vous pouvez facilement convertir la chaîne résultat en une liste en appelant `MEMOTOLIST` (voir Section 16.13.3 [MEMOTOLIST], page 119) avec cette chaîne.

Voir aussi SETLABELS.

### 16.19.4 SETLABELS

`SETLABELS` est utilisé pour placer les entrées d'un champ dans une chaîne.

`(SETLABELS` *`field str`*`)`

Définit les entrées statiques du champ chaîne *field* à partir des entrées listées dans l'argument *str* . L'argument *str* se compose de lignes qui comportent une entrée. Les entrées remplacent celles que vous avez entrées dans la liste de l'éditeur de champ objet (voir Section 15.3.3 [Field object editor], page 73). Notez que cette fonction n'est utile que pour des entrées statiques.

`SETLABELS` renvoie la valeur de l'argument *str*.

Exemple : '`(SETLABELS Table.String "Ma maison\nest\nvotre maison")`' définit les entrées statiques dans la liste d'affichage et indique le champ de la chaîne '`Ma maison`', '`est`', et '`votre maison`'.

Note: vous pouvez facilement convertir une liste d'entrées dans le format requit par une chaîne en appelant LISTTOMEMO sur la liste.

Voir aussi GETLABELS.

## 16.20 Fonctions sur les tables

### 16.20.1 TABLENAME

`TABLENAME` renvoie le nom d'une table.

`(TABLENAME` *`table`*`)`

Renvoie une chaîne contenant le nom de la table indiquée.

Voir aussi FIELDNAME

### 16.20.2 GETORDERSTR

`GETORDERSTR` renvoie un enregistrement trié dans une table.

`(GETORDERSTR` *`table`*`)`

Renvoie l'expression courante de tri pour les données d'une table. Si la table utilise une liste de champs pour le tri, alors la chaîne retournée contiendra les noms des champs séparés par des espaces. Chaque nom de champ est précédé par le signe '`+`' ou '`-`' indiquant un tri croissant ou décroissant.

Si la table est triée par une fonction de comparaison alors le nom de cette fonction est retourné.

Une chaîne vide signifie qu'il n'y a aucun tri.

#### Exemple

Considérez une table '`Personne`' qui est triée par les champs '`Nom`' (croissant), '`Ville`' (croissant), et '`Anniversaire`' (décroissant). Alors, '`(ORDERSTR Personne)`' fournira le résultat de la chaîne `"+Nom +Ville -Anniversaire"`.

Voir aussi SETORDERSTR, REORDER, REORDERALL, GETISSORTED, SETISSORTED, Order, Fonction de comparaison.

### 16.20.3 SETORDERSTR

`SETORDERSTR` place un enregistrement trié dans une table.

`(SETORDERSTR table order)`

Place le tri dans les données de la table selon le champ *Tri*. Le champ *Tri* peut contenir la liste des noms de champs ou le nom d'une fonction de comparaison.

Pour trier en utilisant une liste de champs, le champ *Tri* doit contenir les noms des champs séparés par un nombre d'espaces, tabulations ou une nouvelle ligne. Chaque nom de champ peut être précédé par le signe '`+`' ou a '`-`' indiquant un tri croissant ou décroissant. Si vous omettez ce signe, alors le tri croissant est assumé.

Pour trier en utilisant une fonction de comparaison, le champ *Tri* doit contenir le nom de la fonction.

`SETORDERSTR` renvoie TRUE s'il a pu placer le nouveau tri, NIL autrement, par exemple si un champ inconnu a été indiqué ou si le type du champ n'est pas permis pour le tri. Si vous indiquez NIL pour l'argument *Tri* alors, rien ne se produit et NIL est retourné.

Note : Pour construire le champ tri vous ne devriez pas écrire directement les noms des champs dans un champ parce que quand vous changerez un nom de champ, le champ tri ne sera pas mis à jour. Il vaut mieux utiliser la fonction `FIELDNAME` (voir Section 16.19.1 [FIELDNAME], page 142) qui permet de copier le nom d'un champ dans le champ tri.

#### Exemple

Si l'on considère une table '`Personne`' avec les champs '`Nom`', '`Ville`', et '`Anniversaire`'. alors, '`(SETORDERSTR Personne (SPRINTF "+%s" (FIELDNAME Personne.Nom)))`' placera le tri dans la table '`Personne`' en utilisant le '`Nom`' comme champ de tri (croissant).

Voir aussi GETORDERSTR, REORDER, REORDERALL, GETISSORTED, SETISSORTED, Order, Fonction de comparaison.

### 16.20.4 REORDER

`REORDER` réordonne tous les enregistrements non triés dans le bon ordre.

`(REORDER table)`

Examine tous les enregistrements d'une table de données pour les réordonner et les réinsérer à leur bonne position. Après réinsertion d'un enregistrement réordonné l'état de tri de l'enregistrement est placé sur TRUE, par conséquent l'état de tri de tous les enregistrements renvoyés par `REORDER` est TRUE.

`REORDER` renvoie NIL.

Habituellement vous devez seulement appeler cette fonction lorsque vous employez une fonction de comparaison pour définir le tri d'une table. Les tris définis par une liste de champs sont automatiques, c'est-à-dire, un enregistrement est réordonné automatiquement lorsque c'est nécessaire.

Pour un exemple sur la façon d'utiliser cette fonction, voir Section 16.29.10 [Comparison function], page 161.

Voir aussi REORDERALL, GETORDERSTR, SETORDERSTR, GETISSORTED, SETISSORTED, Order, Fonction de comparaison.

### 16.20.5 REORDERALL

`REORDERALL` réordonne tous les enregistrements d'une table.

`(REORDERALL table)`

Réordonne toutes les données enregistrées d'une table en plaçant l'état de tri de tous les enregistrements sur NIL et en appelant `REORDER` pour tout réordonner.

`REORDERALL` renvoie NIL.

Voir aussi REORDER, GETORDERSTR, SETORDERSTR, GETISSORTED, SETISSORTED, Order, Fonction de comparaison.

### 16.20.6 GETFILTERACTIVE

`GETFILTERACTIVE` renvoie l'état du filtre de la table.

`(GETFILTERACTIVE table)`

Renvoie TRUE si le filtre de la table courante spécifiée est activé et NIL dans les autres cas.

Voir aussi SETFILTERACTIVE, GETFILTERSTR, GETMATCHFILTER.

### 16.20.7 SETFILTERACTIVE

`SETFILTERACTIVE` Place l'état du filtre de la table.

`(SETFILTERACTIVE table bool)`

Place l'état du filtre de la table spécifiée. Si *bool* est différent de NIL alors le filtre est activé, sinon il est désactivé.

`SETFILTERACTIVE` renvoie le nouvel état du filtre. Si vous activez le filtre, le nouvel état attendu ne sera pas définit, mais une erreur se produit et le filtre ne peut pas être activé. Cependant le filtre se désactive toujours avec succès.

Voir aussi GETFILTERACTIVE, SETFILTERSTR, SETMATCHFILTER.

### 16.20.8 GETFILTERSTR

`GETFILTERSTR` renvoie l'expression de l'enregistrement dans le filtre d'une table.

`(GETFILTERSTR table)`

Renvoie l'expression de l'enregistrement pour le filtre d'une table spécifiée dans champs. Un champ vide signifie qu'aucune expression de filtrage n'a été placé pour cette table.

Voir aussi SETFILTERSTR, GETFILTERACTIVE, GETMATCHFILTER.

### 16.20.9 SETFILTERSTR

`SETFILTERSTR` place l'expression de l'enregistrement dans le filtre d'une table.

`(SETFILTERSTR table filter-str)`

Place l'expression de l'enregistrement pour le filtre d'une table spécifiée par l'expression dans l'argument *filter-str* (qui doit être un champ et pas l'expression réelle elle-même !). Si le filtre d'une table donnée est actuellement activé alors, la nouvelle expression de filtrage est appliquée directement à tous les enregistrements et l'état du filtre de tous les enregistrements sont recalculés.

`SETFILTERSTR` renvoie TRUE s'il a pu compiler les données filtrées par le champ de l'expression, sinon NIL est retourné. Notez que vous obtenez seulement le résultat de la

compilation. Si le filtre d'une table donnée est actuellement activé et recalculé tous les états du filtre correspondant aux enregistrements échoueront et ne seront donc pas notés dans le résultat de cette fonction. Pour placer une nouvelle expression de filtrage il est recommandé de procéder de la manière suivante :

```
(SETFILTERACTIVE Table NIL)               ; réussit toujours.
(IF (NOT (SETFILTERSTR Table filter-string))
    (ERROR "ne peut pas placer le champ filtre pour %s!" (TABLENAME Table))
)
(IF (NOT (SETFILTERACTIVE Table TRUE))
    (ERROR "ne peut pas activer le filtre pour %s!" (TABLENAME Table))
)
```

Si SETFILTERSTR est appelé avec une valeur NIL pour l'argument *filter-str* et que rien ne se produit, NIL est retourné.

Exemple : '`(SETFILTERSTR Table "(> Value 0.0)")`'.

Voir aussi GETFILTERSTR, SETFILTERACTIVE, SETMATCHFILTER.

### 16.20.10 RECORDS

RECORDS renvoie le nombre d'enregistrements d'une table.

`(RECORDS` *table*`)`

Renvoie le nombre d'enregistrements d'une table donnée. Vous pouvez apposer une étoile sur le nom d'une table pour compter le nombre d'enregistrements identique au filtre de la table.

Voir aussi RECORD, RECNUM.

### 16.20.11 RECORD

RECORD renvoie l'indicateur d'enregistrement pour le nombre de données enregistrées.

`(RECORD` *table num*`)`

renvoie l'indicateur d'enregistrement vers *num* enregistré dans les données de la table ou NIL si l'enregistrement avec ce nombre n'existe pas. Vous pouvez ajouter une étoile sur le nom d'une table pour obtenir un *num* enregistrement identique à l'enregistrement filtré de la table.

Veuillez noter que les nombres d'enregistrements commencent par 1 et que le numéro d'enregistrement 0 est utilisé pour l'enregistrement initial.

Voir aussi RECORDS, RECNUM.

### 16.20.12 SELECT

SELECT extrait et renvoie diverses données à partir des enregistrements.

```
(SELECT [DISTINCT] exprlist FROM tablelist
        [WHERE where-expr] [ORDER BY orderlist])
```

Ici *exprlist* est soit une simple étoile '`*`' soit une liste d'expressions avec des titres facultatifs séparés par des virgules :

```
exprlist:     * | expr "titre", ...
```

et *tablelist* est une liste de noms de table :

```
tablelist:      table[*] [ident], ...
```

Pour chaque table dans la liste de table vous pouvez indiquer une marque. Ce qui peut être très utile si une table apparaît plus d'une fois dans la liste de table (voir l'exemple de comparaison des âges ci-dessous). Si vous ajoutez une étoile à une table alors seul les enregistrements identiques au filtre actuellement défini dans cette table seront examinés.

orderlist a la syntaxe suivante :

```
orderlist:    expr [ASC | DESC], ...
```

Ici, *expr*, ... peuvent être des expressions arbitraires ou des numéros de champs. Par exemple '`(SELECT Nom FROM ... ORDER BY 1)`' triera le résultat pour le champ '`Nom`'. Vous pouvez spécifier `ASC` ou `DESC` pour un tri croissant ou décroissant. Si aucun d'eux n'est présent, le tri croissant est assumé.

### Comment ça fonctionne

Choisir à partir d'une question construite (mathématique) le résultat produit par toutes les tables dans une liste de table (il examine les séries d'enregistrements dans la *table*, ...) et contrôle où se trouve l'expression (s'il y en a). Si le pointeur de l'expression renvoie TRUE comme résultat (ou s'il n'y a aucun pointeur sur l'expression) alors la liste est une construction dont les éléments sont calculés par une liste d'expression dans la partie sélectionnée. Si vous avez spécifié une simple étoile pour une liste d'expression alors qu'une liste contient les valeurs de tous les champs appartenant aux tables dans une liste de table (excepté les champs virtuels et les boutons).

Le résultat de la question est une liste de listes. La première entrée d'une liste contient le titre d'une chaîne, les autres contiennent les valeurs à partir de la liste dans les enregistrements correspondant.

### Exemples

Voir Section 14.6 [Query examples], page 63, pour quelques exemples utilisant la fonction `SELECT` .

Voir aussi FOR ALL.

## 16.21 Fonctions IHM

Cette section décrit les fonctions pour manipuler les éléments d'une interface graphique.

### 16.21.1 SETCURSOR

`SETCURSOR` place le curseur sur un élément graphique.

```
(SETCURSOR field-or-table)
```

Place le curseur sur un objet graphique représentant un champ ou une table. La fonction ouvre également la fenêtre où le champ/table réside si la fenêtre n'était pas déjà ouverte.

`SETCURSOR` renvoie TRUE si tout va bien (la fenêtre a pu être ouverte) ou NIL en cas d'échec.

Voir aussi SETVIRTUALLISTACTIVE.

### 16.21.2 SETBGPEN

SETBGPEN définit le pinceau d'arrière-plan de l'élément d'IHM.

```
(SETBGPEN field pen)
```

Définit le pinceau pour dessiner l'arrière-plan de l'objet d'IHM indiqué par *field*. Pour *pen*, une valeur entière hexadécimale contenant une couleur au format RVB (rouge, vert, bleu) ou l'une des constantes PEN_* peut être utilisée. Si *pen* vaut NIL alors un arrière-plan par défaut est défini.

SETBGPEN renvoie la valeur du nouveau crayon d'arrière-plan.

Exemple : '`(SETBGPEN Control.Status 0xFF0000)`' définit un fond rouge pour l'élément d'IHM du champ '`Status`' dans la table '`Table`'. Le même effet peut être obtenu avec '`(SETBGPEN Control.Status PEN_RED)`'.

Voir aussi Constantes prédéfinies.

### 16.21.3 GETWINDOWOPEN

GETWINDOWOPEN renvoie l'état d'ouverture d'une fenêtre.

```
(GETWINDOWOPEN field-or-table)
```

Renvoie l'état d'ouverture d'une fenêtre où réside le champ/table.

Voir aussi SETWINDOWOPEN.

### 16.21.4 SETWINDOWOPEN

SETWINDOWOPEN ouvre ou ferme une fenêtre.

```
(SETWINDOWOPEN field-or-table open)
```

Ouvre ou ferme la fenêtre dans laquelle réside le champ/table. Si *open* n'est pas NIL alors la fenêtre est ouverte, sinon elle est fermée. Vous ne pouvez pas fermer la fenêtre principale d'un projet.

SETWINDOWOPEN renvoie le nouvel état d'ouverture de la fenêtre.

Voir aussi GETWINDOWOPEN.

### 16.21.5 GETVIRTUALLISTACTIVE

GETVIRTUALLISTACTIVE renvoie l'index de la ligne active d'un champ virtuel qui utilise une '`List`' pour l'affichage.

```
(GETVIRTUALLISTACTIVE virtual-field)
```

Renvoie l'index (commençant par 1) de la ligne active courante avec le champ Virtuel spécifié. Si l'élément d'IHM de *virtual-field* n'est pas visible, s'il n'emploie pas de '`Liste`' pour son affichage, ou si aucune ligne n'est activée, alors NIL est retourné.

Voir aussi SETVIRTUALLISTACTIVE.

### 16.21.6 SETVIRTUALLISTACTIVE

SETVIRTUALLISTACTIVE Modifie la ligne active d'un champ Virtuel utilisant une '`Liste`' pour son affichage.

```
(SETVIRTUALLISTACTIVE virtual-field num)
```

Modifie la ligne active du champ Virtuel sur la *num*eme ligne (commençant à 1). Renvoie *num*, ou NIL si l'élément d'IHM de *virtual-field* n'est pas visible, n'utilise pas de '`Liste`'

pour son affichage ou si *num* est hors limite (inférieure à 1 ou plus grand que le nombre de lignes).

`SETVIRTUALLISTACTIVE` ne place pas le curseur sur l'élément d'IHM du champ, utilisez pour cela `SETCURSOR` (voir Section 16.21.1 [SETCURSOR], page 147).

Voir aussi GETVIRTUALLISTACTIVE, SETCURSOR.

## 16.22 Fonctions projets

Cette section énumère les fonctions traitant des projets.

### 16.22.1 PROJECTNAME

`PROJECTNAME` renvoie le nom du projet.

```
(PROJECTNAME)
```

`PROJECTNAME` renvoie le nom du projet en cours dans une chaîne ou NIL si aucun nom n'a encore été défini. Le nom du project est le chemin du répertoire-projet du système de fichier.

Voir aussi CHANGES.

### 16.22.2 PREPARECHANGE

`PREPARECHANGE` Préparer le projet à une modification.

```
(PREPARECHANGE)
```

Cette commande pose un verrou de modification sur le projet. Ceci est utile lors de l'accès à un projet avec plusieurs instances de BeeBase s'exécutant éventuellement sur différents ordinateurs. Une fois qu'un verrou de modification a été obtenu, aucune autre instance BeeBase ne peut en obtenir un tant que le verrou n'est pas libéré. Le verrou de modification est libéré si le programme de projet revient sans effectuer réellement de modification ou après l'enregistrement du projet. Pour plus d'informations sur le partage d'un projet, voir Section 6.1 [File format], page 30.

`PREPARECHANGE` renvoie NIL en cas de succès. Si l'obtention du verrou de modification échoue, le programme se ferme avec un message d'erreur approprié.

Voir aussi CHANGES.

### 16.22.3 CHANGES

`CHANGES` renvoie le nombre de modifications du projet en cours.

```
(CHANGES)
```

Renvoie un nombre entier contenant le nombre de modifications depuis la dernière opération de sauvegarde du projet en cours.

Voir aussi PROJECTNAME.

### 16.22.4 GETADMINMODE

`GETADMINMODE` tells whether the current project is in admin mode `GETADMINMODE` indique si le projet courant est en mode administrateur ou utilisateur.

`GETADMINMODE` indique si le projet en cours est en mode administrateur `GETADMINMODE` indique si le projet courant est en mode administrateur ou utilisateur.

```
(GETADMINMODE)
```

Retourne TRUE si le projet courant est en mode administrateur, NIL sinon.

Voir aussi SETADMINMODE, ADMINPASSWORD, onAdminMode.

### 16.22.5 SETADMINMODE

`SETADMINMODE` bascule le projet courant vers le mode administrateur ou utilisateur.

```
(SETADMINMODE admin)
```

Si *admin* est NIL alors le projet courant est basculé en mode utilisateur, dans le cas contraire il passe en mode administrateur. Notez qu'il n'y a pas de fenêtre d'identification lors du passage du mode utilisateur vers le mode administrateur en utilisant cette fonction.

Retourne TRUE si le projet a été basculé en mode administrateur, ou NIL s'il est basculé en mode utilisateur.

Voir aussi GETADMINMODE, ADMINPASSWORD, onAdminMode, démo `Users.bbs`.

### 16.22.6 ADMINPASSWORD

`ADMINPASSWORD` obtient le mot de passe administrateur sous forme de chaîne de hachage SHA1.

```
(ADMINPASSWORD)
```

Retourne une chaîne représentant la valeur de hachage SHA1 du mot de passe administrateur du projet courant. Si aucun mot de passe administrateur n'a été configuré, NIL est retourné.

Voir aussi GETADMINMODE, SETADMINMODE, SHA1SUM, démo `Users.bbs`.

## 16.23 Fonctions système

Cette section énumère les fonctions faisant appel au système d'exploitation.

### 16.23.1 EDIT

`EDIT` lance un éditeur externe.

```
(EDIT filename)
```

Démarrez un éditeur externe pour éditer un fichier spécifié. L'éditeur externe peut être placé dans le menu '`Préférences - Configurer l'éditeur externe`' (voir Section 7.1.2 [External editor], page 34). `EDIT` démarre l'éditeur externe synchroniquement, ce qui veut dire qu'il attend jusqu'à ce que l'utilisateur sorte de l'éditeur.

`EDIT` renvoie le code d'erreur de l'éditeur externe sous forme de nombre entier.

Voir aussi EDIT*, VIEW, SYSTEM.

### 16.23.2 EDIT*

`EDIT*` est la version étoilée de `EDIT` et a le même effet que `EDIT` (voir Section 16.23.1 [EDIT], page 150). La seule différence est que `EDIT*` démarre un éditeur externe de façon asynchrone, ainsi la fonction rend la main immédiatement.

`EDIT*` renvoie 0 en cas de succès lors du démarrage de l'éditeur, sinon il renvoie la valeur d'un nombre entier différent de zéro représentant un code d'erreur système spécifique.

Voir aussi EDIT, VIEW*, SYSTEM*.

### 16.23.3 VIEW

`VIEW` lance la visionneuse externe.

```
(VIEW filename)
```

Démarre la visionneuse externe pour afficher le fichier spécifié. La visionneuse externe peut être placée dans menu '`Préférences - Configurer la visionneuse`' (voir Section 7.1.3 [External viewer], page 35). `VIEW` démarre la visionneuse externe de façon synchrone, c'est-à-dire qu'elle attend jusqu'à ce que l'utilisateur sorte de la visionneuse. Notez que sur quelques systèmes, l'appel pourrait immédiatement être renvoyé si un exemple de la visionneuse fonctionne déjà.

`VIEW*` renvoie le code d'erreur renvoyé par la visionneuse externe sous forme de nombre entier.

Voir aussi VIEW*, EDIT, SYSTEM.

### 16.23.4 VIEW*

`VIEW*` est la version étoilée de `VIEW` et a les mêmes effets que `VIEW` (voir Section 16.23.3 [VIEW], page 151). La seule différence est que `VIEW*` démarre la visionneuse externe de façon asynchrone, ainsi la fonction rend la main immédiatement.

`VIEW*` renvoie 0 s'il réussit à démarrer la visionneuse, sinon il renvoie la valeur d'un nombre entier différent de zéro représentant un code d'erreur système spécifique.

Voir aussi VIEW, EDIT*, SYSTEM*.

### 16.23.5 SYSTEM

`SYSTEM` appelle un programme externe.

```
(SYSTEM fmt [arg ...])
```

Appelle un programme externe. La ligne de commande pour appeler le programme est spécifiée par *fmt* et des arguments facultatifs comme dans la fonction `SPRINTF` (voir Section 16.12.34 [SPRINTF], page 116). Pour interpréter la ligne de commande, le shell du système est utilisé (/bin/sh sur Linux, ShellExecute sur Windows, le shell utilisateur sur Amiga). `SYSTEM` attend jusqu'à ce que le programme appelé soit terminé.

`SYSTEM` renvoie le code retour de la commande exécutée sous forme de nombre entier.

Voir aussi SYSTEM*, EDIT, VIEW.

### 16.23.6 SYSTEM*

`SYSTEM*` est la version étoilée de `SYSTEM` et a les mêmes effets que `SYSTEM` (voir Section 16.23.5 [SYSTEM], page 151). La seule différence est que `SYSTEM*` exécute la ligne de commande de façon asynchrone, ainsi la fonction rend la main immédiatement.

`SYSTEM*` renvoie 0 s'il réussit à lancer l'exécution de la ligne de commande, sinon il renvoie un nombre entier différent de zéro représentant un code d'erreur spécifique au système.

Voir aussi SYSTEM, EDIT*, VIEW*.

### 16.23.7 STAT

`STAT` examine un nom de fichier.

`(STAT` *`filename`*`)`

Vérifie si le fichier dont le nom a été indiqué existe dans le système. `STAT` renvoie NIL si le fichier ne peut pas être trouvé, 0 si le fichier existe et est un répertoire, et un nombre entier supérieur à 0 si le fichier existe et est un fichier normal.

### 16.23.8 TACKON

`TACKON` crée le chemin d'accès.

`(TACKON` *`dirname`* `[`*`component ...`*`])`

Fusionne *dirname* et tous les composants de [*component* ...] pour obtenir un chemin d'accès. `TACKON` sait traiter les caractères spéciaux utilisés comme séparateurs à la fin de chaque élément. Il renvoie le chemin d'accès dans un champ ou NIL si aucun des arguments est NIL. Notez que `TACKON` n'effectue aucun contrôle sur l'existence ou non du chemin d'accès résultant.

Exemple : '`(TACKON "Sys:System" "CLI")`' donne "Sys:System/CLI".

Voir aussi FILENAME, DIRNAME.

### 16.23.9 FILENAME

`FILENAME` extrait le nom de fichier à partir du chemin d'accès.

`(FILENAME` *`path`*`)`

Extrait le dernier composant du chemin d'accès spécifié. Aucune vérification n'est faite si le *path* spécifié se rapporte réellement à un fichier, ainsi il est également possible d'utiliser `FILENAME` pour obtenir le nom d'un sous-répertoire. `FILENAME` renvoie son résultat sous forme de chaîne ou NIL si *path* est NIL.

Exemple : '`(FILENAME "Sys:System/CLI")`' donne "CLI".

Voir aussi DIRNAME, TACKON.

### 16.23.10 DIRNAME

`DIRNAME` extrait une partie correspondant au répertoire depuis un chemin d'accès.

`(DIRNAME` *`path`*`)`

Extrait la partie correspondant au répertoire du chemin d'accès spécifié. Aucune vérification n'est faite si *path* se rapporte effectivement à un fichier, ainsi il est également possible d'utiliser `DIRNAME` pour obtenir le répertoire parent. `DIRNAME` renvoie son résultat sous forme de chaîne ou NIL si *path* est NIL.

Exemple : '`(DIRNAME "Sys:System/CLI")`' donne "Sys:System".

Voir aussi FILENAME, TACKON.

### 16.23.11 MESSAGE

`MESSAGE` affiche un message à l'utilisateur.

`(MESSAGE` *`fmt`* `[`*`arg ...`*`])`

Positionne le titre de la fenêtre pause/abort (si elle est ouverte). Le titre est créé à partir de *fmt* et des arguments facultatifs comme avec la fonction `SPRINTF` (voir Section 16.12.34 [SPRINTF], page 116).

`MESSAGE` renvoie le champ titre formaté.

Exemple : '`(MESSAGE "6 * 7 = %i" (* 6 7))`'.

Voir aussi PRINT, PRINTF.

### 16.23.12 COMPLETEMAX

`COMPLETEMAX` spécifie le nombre maximum d'étapes de progression.

`(COMPLETEMAX steps)`

Spécifie le nombre maximum d'étapes pour montrer à l'utilisateur la progression de votre programme BeeBase. Le nombre d'étapes par défaut (si vous n'appelez pas cette fonction) est de 100. La valeur de l'argument *steps* doit être un nombre entier. Si *steps* est NIL ou 0 alors aucune barre de progression n'est affichée. La barre de progression fait partie de la fenêtre pause/arrêt et surgit après un bref délai en exécutant un programme BeeBase.

`COMPLETEMAX` renvoie son argument *steps*.

Voir aussi COMPLETEADD, COMPLETE.

### 16.23.13 COMPLETEADD

`COMPLETEADD` incrémente l'état de progression.

`(COMPLETEADD add)`

Ajoute le nombre entier contenu dans *add* à la valeur courante de la progression. La valeur initiale de progression est placée sur 0. Si *add* est NIL alors la valeur de progression est remise à 0 et aucune barre de progression n'est montrée.

`COMPLETEADD` renvoie son argument *add*.

#### Exemple :

```
(SETQ num ...)
(COMPLETEMAX num)
(DOTIMES (i num)
    (COMPLETEADD 1)
)
```

Voir aussi COMPLETEMAX, COMPLETE.

### 16.23.14 COMPLETE

`COMPLETE` modifie l'état de progression.

`(COMPLETE cur)`

Positionne la valeur courante de progression sur la valeur entière *cur*. Si *cur* est NIL ou 0 alors aucune barre de progression n'est montrée.

`COMPLETE` renvoie son argument *cur*.

**Exemple :**

```
(COMPLETE 10)
...
(COMPLETE 50)
...
(COMPLETE 100)
```

Voir aussi COMPLETEMAX, COMPLETEADD.

### 16.23.15 GC

`GC` force le passage du « ramasse-miettes ».

```
(GC)
```

Force le passage du « ramasse-miettes » et renvoie NIL. Normalement le « ramasse-miettes » est exécuté automatiquement de manière régulière.

### 16.23.16 PUBSCREEN

`PUBSCREEN` retourne le nom de l'écran publique.

```
(PUBSCREEN)
```

Sur Amiga, `PUBSCREEN` retourne le nom de l'écran public sur lequel est affiché BeeBase, ou NIL si l'écran n'est pas public.

Sur tous les autres systèmes, `PUBSCREEN` retourne NIL.

## 16.24 Les variables prédéfinies

BeeBase connaît quelques variables globales prédéfinies.

Dans la version en cours il existe seulement une variable globale : `stdout` (voir Section 16.17.3 [stdout], page 134).

## 16.25 Constantes prédéfinies

Les constantes prédéfinies suivantes peuvent être employées dans toute expression pour la programmation.

| Nom | Type | Valeur | Commentaire |
|---|---|---|---|
| NIL | tous | NIL | |
| TRUE | booléen | TRUE | |
| RESET | texte | "\33c" | |
| NORMAL | texte | "\33[0m" | |
| ITON | texte | "\33[3m" | |
| ITOFF | texte | "\33[23m" | |
| ULON | texte | "\33[4m" | |
| ULOFF | texte | "\33[24m" | |
| BFON | texte | "\33[1m" | |
| BFOFF | texte | "\33[22m" | |
| ELITEON | texte | "\33[2w" | |
| ELITEOFF | texte | "\33[1w" | |

```
CONDON          texte           "\33[4w"
CONDOFF         texte           "\33[3w"
WIDEON          texte           "\33[6w"
WIDEOFF         texte           "\33[5w"
NLQON           texte           "\33[2\"z"
NLQOFF          texte           "\33[1\"z"
INT_MAX         entier          2147483647      Valeur entière maximum
INT_MIN         entier          -2147483648     Valeur entière minimum
HUGE_VAL        réel            1.797693e+308   Valeur réelle absolue maximum
PI              réel            3.14159265359
OSTYPE          texte           <Type OS>       "Unix", "Windows" ou "Amiga"
OSVER           entier          <Version OS>
OSREV           entier          <Révision OS>
BBVER           entier          <Version BeeBase>
BBREV           entier          <Révision BeeBase>
LANGUAGE        texte           dépendant       Langue par défaut
SEEK_SET        entier          cf. stdio.h     Position en début de fichier
SEEK_CUR        entier          cf. stdio.h     Position courante
SEEK_END        entier          cf. stdio.h     Position en fin de fichier
```

Voir Section 16.4.7 [Constants], page 87, pour plus d'informations sur les constantes. Pour définir vos propres constantes, utilisez l'instruction du préprocesseur `#define` (voir Section 16.3.1 [#define], page 82).

## 16.26 Paramètres fonctionnels

Vous pouvez passer une fonction comme argument à une autre fonction. C'est utile pour définir des fonctions évoluées, comme par exemple pour trier ou construire une liste.

Pour appeler une fonction qui a été passée en argument vous devez utiliser la fonction `FUNCALL` (voir Section 16.6.8 [FUNCALL], page 93).

### Exemple :

```
(DEFUN map (l fun)              # arguments: list and function
    (LET (res)                  # local variable res, initialized with NIL
        (DOLIST (i l)           # for all items one by one
            (SETQ res
                (CONS (FUNCALL fun i) res)       # calls function and
            )                                    # build new list
        )
        (REVERSE res)           # we need to reverse the new list
    )
)
```

Vous pouvez maintenant employer la fonction map par exemple pour incrémenter tous les articles d'une liste de nombres entiers :

'`(map (LIST 1 2 3 4) 1+)`' donne ( 2 3 4 5 ).

Voir aussi FUNCALL, APPLY, MAPFIRST.

## 16.27 Spécificateurs de type

Il est possible d'indiquer le type d'une variable en ajoutant un spécificateur de type derrière son nom. Les spécificateurs de type suivants existent :

| **Spécificateur** | **Description** |
|---|---|
| `:INT` | pour les entiers |
| `:REAL` | pour les réels |
| `:STR` | pour les chaînes |
| `:MEMO` | pour les mémos |
| `:DATE` | pour les dates |
| `:TIME` | pour les heures |
| `:LIST` | pour les listes |
| `:FILE` | pour les descripteurs de fichier |
| `:FUNC` | pour les fonctions de n'importe quel type |
| `:`*table* | pour les pointeurs d'enregistrement sur *table* |

un spécificateur de type s'accole au nom de variable comme dans l'exemple suivant :

```
(LET (x:INT (y:REAL 0.0) z) ...)
```

L'exemple définit trois nouvelles variables '`x`', '`y`' et '`z`', où '`x`' est de type entier et initialisé avec NIL, '`y`' est de type réel et initialisé avec 0.0, et '`z`' est une variable sans type initialisé avec NIL.

L'avantage des spécificateurs de type est que le compilateur peut détecter plus d'erreurs de typage, p. ex. si vous avez une fonction :

```
(DEFUN foo (x:INT) ...)
```

et que vous l'appelez avec '`(foo "bar")`', le compilateur produira un message d'erreur. Cependant, si vous appelez '`foo`' avec une valeur non typée, p. ex. '`(foo (FIRST list))`' alors aucune vérification d'erreur de typage ne peut être faite lors de la compilation puisque le type de '`(FIRST list)`' est inconnu à ce moment.

Pour des raisons de performance, aucune vérification de typage n'est actuellement faite lors de l'exécution. Cela pourrait être implémenté, mais ajouterait une légère surcharge inutile puisque de toute façon un mauvais type produira tôt ou tard une erreur de typage.

Les spécificateurs de type pour les pointeurs d'enregistrements ont une autre fonction utile. Si vous étiquetez une variable comme pointeur d'enregistrement vers une table alors vous pourrez accéder à tous les champs de cette table en employant le nom de la variable au lieu de celui de la table dans le chemin d'accès au champ. Par exemple si vous avez une table '`Foo`' avec un champ '`Bar`', et que vous définissez une variable '`foo`' comme :

```
(LET (foo:Foo))
```

alors vous pourrez afficher le champ '`Bar`' du troisième enregistrement en utilisant :

```
(SETQ foo (RECORD Foo 3)) (PRINT foo.Bar)
```

Notez que dans une expression select-from-where, les variables définies dans la liste ont automatiquement un type pointeur d'enregistrement vers la table correspondante.

## 16.28 Sémantique des expressions

La sémantique des expressions est très importante pour la compréhension des programmes. Cette section énumère la sémantique selon des expressions syntaxiques.

(*func* [*expr* ...])
: Évalue *expr* ... puis appelle la fonction *func* (appel par valeur). Renvoie la valeur de retour de la fonction appelée. Dans BeeBase il existe quelques fonctions non strictes, par exemple. `AND`, `OR` et `IF`. Ces fonctions peuvent ne pas évaluer toutes les expressions. Pour plus d'informations sur les fonctions non strictes, voir Section 16.4.2 [Lisp syntax], page 85, Section 16.9.1 [AND], page 102, Section 16.9.2 [OR], page 103, et Section 16.6.10 [IF], page 94.

([*expr* ...])
: Évalue *expr* ... et renvoie la valeur de la dernière expression (voir Section 16.6.1 [PROGN], page 91). Une expression vide `()` s'évalue en NIL.

*Table*
: Renvoie le pointeur d'enregistrement du programme pour la table.

*Table**
: Renvoie le pointeur d'enregistrement de l'interface pour la table de données.

*FieldPath*
: Renvoie le contenu du champ spécifié. Le chemin du champ spécifie quel enregistrement est utilisé pour extraire la valeur du champ. Par exemple '`Table.Champ`' utilise l'enregistrement du programme '`Table`' pour extraire la valeur du champ, '`Table.ChampRéférence.Champ`' utilise l'enregistrement du programme '`Table`' pour extraire la valeur du champ de référence (qui est un pointeur d'enregistrement) et utilise alors cet enregistrement pour extraire la valeur '`Champ`'.

*var*
: Renvoie le contenu de la variable globale ou locale *var*. Les variables globales peuvent être définies avec `DEFVAR` (voir Section 16.5.3 [DEFVAR], page 90), et les variables locales par exemple avec `LET` (voir Section 16.6.3 [LET], page 91).

*var.FieldPath*
: Utilise le pointeur d'enregistrement *var* pour déterminer la valeur du champ spécifié.

## 16.29 Déclenchement de fonction

Pour exécuter automatiquement des programmes BeeBase vous pouvez spécifier des fonctions « déclencheurs » sur les projets, les tables et les champs qui sont appelés dans des cas spécifiques. Cette section énumère toutes les possibilités de déclenchement.

### 16.29.1 onOpen

Après l'ouverture d'un projet, BeeBase recherche dans le programme du projet une fonction appelée `onOpen`. Si une telle fonction existe alors elle est appelée sans aucun argument.

### Exemple

```
(DEFUN onOpen ()
    (ASKBUTTON NIL "Merci de m'ouvrir !" NIL NIL)
)
```

Voir aussi onClose, onAdminMode, onChange, la démo `Trigger.bbs`.

## 16.29.2 onReload

Semblable à `onOpen` (voir Section 16.29.1 [onOpen], page 157) BeeBase appelle la fonction `onReload` lors du rechargement d'un projet, par ex. en sélectionnant l'élément de menu '`Project - Reload`'. La fonction est appelée sans argument.

Cependant, `onReload` n'est appelé que s'il n'y a pas de modifications structurelles du projet, c'est-à-dire qu'aucune table ou champ n'a été modifié, ajouté ou supprimé, qu'aucune modification n'a été apportée au programme du projet et qu'aucune autre modification n'est présente. comme un changement structurel. Dans un tel cas, un rechargement est considéré comme une réouverture du projet, et la fonction `onOpen` est appelée à la place.

Voir aussi onOpen, onClose, onAdminMode, onChange.

## 16.29.3 onClose

Avant la fermeture d'un projet, BeeBase recherche dans le programme du projet une fonction appelée `onClose`. Si une telle fonction existe alors elle est appelée sans argument. Dans la version courante le résultat de ce déclencheur est ignoré et le projet est fermé sans se soucier de sa valeur de retour.

Si vous effectuez des modifications du projet dans la fonction `onClose`, alors BeeBase vous demandera d'abord de sauvegarder le projet avant de le fermer réellement. Si vous utilisez le menu '`Projet - Sauver & Fermer`' pour la fermeture d'un projet, le déclenchement sera appelé avant de sauver le projet, ainsi les changements seront sauvés automatiquement.

### Exemple

```
(DEFUN onClose ()
    (ASKBUTTON NIL "Au revoir !" NIL NIL)
)
```

Voir aussi onOpen, onChange, démo `Trigger.bbs`.

## 16.29.4 onAdminMode

À chaque fois qu'un projet passe en mode administrateur ou utilisateur et qu'une fonction appellée `onAdminMode` existe dans un programme projet alors cette fonction est appellée. La fonction reçoit un argument *admin* indiquant si le projet est en mode administrateur (*admin* n'est pas NIL) ou en mode utilisateur (*admin* est NIL).

### Exemple

```
(DEFUN onAdminMode (admin)
    (IF admin
        (ASKBUTTON NIL "Maintenant en mode administrateur" NIL NIL)
        (ASKBUTTON NIL "Revenu en mode utilisateur" NIL NIL)
```

```
    )
)
```

Voir aussi onOpen, onChange, SETADMINMODE, la démo `Users.bbs`.

### 16.29.5 onChange

À chaque fois que l'utilisateur fait des changements sur un projet ou après avoir enregistré un projet, BeeBase recherche dans le programme du projet une fonction appelée `onChange`. Si une telle fonction existe, elle est exécutée sans argument. Ce qui peut être utilisé pour compter les changements faits par l'utilisateur sur un projet.

#### Exemple

```
(DEFUN onChange ()
    (SETQ Control.NumChanges (CHANGES))
)
```

Dans l'exemple ci-dessus '`Control.NumChanges`' pourrait être un champ virtuel quelque part dans une table de type '`Exactement un enregistrement`' pour afficher le nombre de modifications du projet.

Voir aussi onOpen, onClose, onAdminMode, la démo `Trigger.bbs`.

### 16.29.6 logLabel

Lors de la création d'une nouvelle entrée pour le journal du projet, BeeBase recherche dans le programme du projet une fonction portant le nom `logLabel`. S'il existe, il appelle cette fonction sans aucun argument. L'expression renvoyée est convertie en chaîne et utilisée pour le champ '`_Label`' de la nouvelle entrée de journal (voir Section 8.5 [Set log label], page 42).

#### Example

```
(DEFUN logLabel ()
    Control.CurrentUser.Name
)
```

L'exemple ci-dessus est tiré du projet `Users.bbs`. Ici, l'étiquette est le nom de l'utilisateur actuel qui a ouvert le projet. Cela signifie que toutes les modifications effectuées par l'utilisateur actuel sont marquées avec son nom. Ainsi, il est possible ultérieurement d'identifier quel utilisateur a effectué quel changement.

Voir aussi onChange, démo `Users.bbs`.

### 16.29.7 mainWindowTitle

Si le programme d'un projet contient une fonction avec le nom `mainWindowTitle` alors le résultat de cette fonction est utilisé pour définir le titre de la fenêtre principale du projet. La fonction est appelée sans aucun argument et doit renvoyer une chaîne. Le titre de la fenêtre est automatiquement recalculé à chaque fois qu'un champ utilisé dans la fonction change.

Si `mainWindowTitle` n'existe pas, le titre de la fenêtre affiche le nom du projet.

Notez que, dans tous les cas, BeeBase ajoute au titre un caractère '`*`' chaque fois que le projet contient des modifications non enregistrées.

Voir aussi démo `Trigger.bbs`.

### 16.29.8 Déclencheur de création

Quand l'utilisateur veut allouer un nouvel enregistrement en sélectionnant l'un des éléments du menu '`Nouvel enregistrement`' ou '`Dupliquer l'enregistrement`' et qu'un déclencheur de '`Création`' a été spécifié pour cette table, cette fonction de déclenchement est exécutée. Le déclencheur de '`Création`' peut être spécifié dans la fenêtre de saisie des tables (voir Section 15.1.1 [Creating tables], page 65).

Le déclencheur reçoit NIL ou un pointeur d'enregistrement en tant que premier et seul argument. NIL signifie que l'utilisateur veut allouer un nouvel enregistrement tandis qu'un pointeur d'enregistrement signifie que l'utilisateur veut dupliquer cet enregistrement. Si le déclencheur possède plus d'un argument alors ceux-ci seront initialisés avec NIL. Le déclencheur doit allouer le nouvel enregistrement en appelant la fonction `NEW` (voir Section 16.18.1 [NEW], page 137). Le résultat retourné par le déclencheur sera examiné et s'il renvoie un pointeur d'enregistrement alors l'enregistrement sera affiché.

Le déclencheur de '`Création`' est également exécuté lorsqu'un programme BeeBase appelle la fonction `NEW*` (voir Section 16.18.2 [NEW*], page 138).

#### Exemple de déclencheur de création

```
(DEFUN nouvelEnregistrement (init)
    (PROG1                      ; pour renvoyer le résultat de NEW
        (NEW Table init)
        ...
    )
)
```

Voir aussi NEW, NEW*, Déclencheur de suppression.

### 16.29.9 Déclencheur de suppression

Lorsque l'utilisateur veut supprimer un enregistrement en sélectionnant l'élément du menu '`Supprimer l'enregistrement`' et qu'un déclencheur de '`Suppression`' a été spécifié pour cette table, alors cette fonction de déclenchement est exécutée. Le déclencheur de '`Suppression`' peut être spécifié dans la fenêtre de saisie des tables (voir Section 15.1.1 [Creating tables], page 65).

Le déclencheur reçoit un argument booléen comme seul argument. Si l'argument est différent de NIL alors la fonction demande à l'utilisateur s'il veut vraiment supprimer l'enregistrement. Si c'est le cas, le déclencheur doit appeler `DELETE` (voir Section 16.18.3 [DELETE], page 138) pour supprimer l'enregistrement.

Le déclencheur de '`Suppression`' est également appelé lorsqu'un programme BeeBase appelle la fonction `DELETE*` (voir Section 16.18.4 [DELETE*], page 139).

#### Exemple de déclencheur de suppression

```
(DEFUN supprimerEnregistrement (confirmation)
    (DELETE Table confirmation)
)
```

Voir aussi DELETE, DELETE*, Déclencheur de création.

### 16.29.10 Fonction de comparaison

Pour trier les enregistrements d'une table vous pouvez utiliser une fonction de comparaison. Voir Section 11.4 [Changing orders], page 52, pour savoir comment doit être spécifiée une telle fonction pour une table. La fonction prend deux pointeurs d'enregistrements en arguments et renvoie une valeur entière reflétant l'ordre de tri des deux enregistrements. La fonction de comparaison doit renvoyer une valeur inférieure à 0 si son premier argument est plus petit que le second, 0 s'ils sont équivalents et une valeur supérieure à 0 si le premier argument est plus grand que le second.

Par exemple, si vous avez une table '`Personnes`' avec un champ de type chaîne '`Nom`' alors vous pourriez utiliser la fonction suivante pour comparer deux enregistrements :

```
(DEFUN cmpPersonnes (rec1:Personnes rec2:Personnes)
    (CMP rec1.Nom rec2.Nom)
)
```

Ceci triera tous les enregistrements selon le champ '`Nom`' en utilisant la comparaison de chaîne sensible à la casse. Notez qu'en utilisant une liste de champs vous ne pourriez pas obtenir le même tri car, dans le cas des listes de champs, une comparaison de chaînes sensible à la casse est effectuée.

En utilisant une fonction de comparaison vous pouvez définir des relations d'ordre très complexes. Faîtes attention à ne pas créer de fonctions récursives qui s'appelleraient elles-mêmes. BeeBase arrêtera l'exécution du programme et vous affichera un message d'erreur si vous essayez de faire cela. En outre, vous ne devez pas utiliser de commandes à effets de bord, par exemple positionner la valeur d'un champ.

En utilisant une fonction de comparaison, BeeBase ne sait pas toujours lorsqu'il doit réordonner les enregistrements. Par exemple si l'on considère dans l'exemple précédent une nouvelle table '`Jouets`' possédant un champ de type chaîne '`Nom`' et une référence '`Propriétaire`' vers '`Personnes`' et la fonction suivante pour comparer les enregistrements :

```
(DEFUN cmpJouets (rec1:Jouets rec2:Jouets)
    (CMP* rec1.Proprietaire rec2.Propriétaire)
)
```

Cette fonction utilise le tri des '`Personnes`' pour déterminer l'ordre de tri des enregistrements, par conséquent les enregistrements de '`Jouets`' sont classés selon le tri des '`Personnes`'.

Maintenant si l'utilisateur modifie un enregistrement de la table '`Personnes`' et que la position (dans le tri) de cet enregistrement change, alors tous les enregistrements de '`Jouets`' se rapportant à cet enregistrement ont besoin d'être réordonnés. Cependant, BeeBase ne connaît pas cette dépendance.

En plus d'utiliser le menu '`Table - Réordonner tous les enregistrements`' pour retrier les enregistrements de la table '`Jouets`', vous pouvez mettre en place une réorganisation automatique en spécifiant le déclencheur de champ suivant sur le champ '`Nom`' de la table '`Personnes`' :

```
(DEFUN setNom (nouvelleValeur)
    (SETQ Personnes.Nom nouvelleValeur)
    (FOR ALL Jouets WHERE (= Jouets.Proprietaire Personnes) DO
```

```
        (SETISSORTED Jouets NIL)
    )
    (REORDER Jouets)
)
```

La fonction supprime l'information de tri pour tous les enregistrements se rapportant à l'enregistrement courant de la table '`Personnes`' et réordonné alors tous les enregistrements non triés de la table '`Jouets`'.

Voir aussi Order, CMP, GETISSORTED, SETISSORTED, REORDER, REORDERALL, GETORDERSTR, SETORDERSTR, PRINT, PRINTF, la démo `Order.bbs`.

### 16.29.11 Déclencheur de champ

Dans la fenêtre de création de champs (voir Section 15.2.1 [Creating fields], page 67) vous pouvez définir une fonction qui sera déclenchée toutes les fois que l'utilisateur voudra modifier le contenu d'un champ : le déclencheur de champ.

Si vous avez défini une telle fonction pour un champ et que l'utilisateur modifie la valeur de ce champ alors le contenu de l'enregistrement ne sera pas automatiquement mis à jour avec la nouvelle valeur. Au lieu de cela la valeur est passée en paramètre au déclencheur. Le déclencheur peut alors vérifier la valeur et éventuellement la refuser. Pour stocker la valeur dans un enregistrement vous devez utiliser la fonction `SETQ`.

Le déclencheur doit renvoyer le résultat de l'appel à `SETQ` ou l'ancienne valeur du champ s'il décide de refuser la nouvelle.

Le déclencheur est également appelé lorsqu'un programme BeeBase fait appel à la fonction `SETQ*` (voir Section 16.6.5 [SETQ*], page 92) pour positionner la valeur d'un champ.

#### Exemple de déclencheur de champ

```
(DEFUN setMontant (montant)
    (IF une-expression
        (SETQ Table.Montant montant)
        (ASKBUTTON NIL "Valeur Invalide !" NIL NIL)
    )
    Table.Montant                       ; retourne la valeur courante
)
```

Voir aussi SETQ*

### 16.29.12 Programmation de champs virtuels

Dans BeeBase les champs virtuels sont des champs spéciaux qui calculent leur valeur à chaque fois que c'est nécessaire. Par exemple si vous passez à un autre enregistrement en cliquant sur le bouton flèche dans l'onglet de contrôle d'une table. Le champ Virtuel de cette table sera alors automatiquement recalculé et affiché (les réglages appropriés pour le champ Virtuel étant fournis, voir Section 15.3.3 [Field object editor], page 73). Pour calculer la valeur du champ, le déclencheur de '`Calcul`' est appelé. Ce déclencheur peut être spécifié dans la fenêtre de saisie de champ (voir Section 15.2.2 [Type specific settings], page 67). La valeur renvoyée par ce déclencheur définit la valeur du champ Virtuel. Si vous ne spécifiez aucun déclencheur de '`Calcul`' pour un champ Virtuel, alors la valeur du champ est NIL

Vous pouvez également déclencher le calcul d'un champ Virtuel en y accédant simplement à partir d'un programme BeeBase. Ainsi par exemple, si vous avez un bouton qui doit calculer la valeur d'un champ Virtuel, vous devrez seulement spécifier une fonction pour le bouton comme celle qui suit :

```
(DEFUN buttonHook ()
    champ-virtuel
)
```

Vous pouvez également donner à un champ Virtuel n'importe quelle valeur en utilisant la fonction `SETQ` :

```
(SETQ champ-virtuel expr)
```

Cependant si vous accédez au champ Virtuel après l'appel de `SETQ`, la valeur du champ Virtuel sera recalculée.

Il n'y a pas de mécanisme de cache de la valeur d'un champ Virtuel parce qu'il n'est pas facile de savoir quand la valeur doit ou ne doit pas être recalculée. Par conséquent, il est préférable d'accéder rarement aux champs virtuels et d'en mémoriser la valeur dans des variables locales pour un usage ultérieur.

Pour un exemple sur la façon d'utiliser les champs virtuels veuillez consulter la démo `Movie.bbs` .

Voir aussi Champs virtuels, la démo `Movie.bbs`.

### 16.29.13 Fonction de calcul d'activation

Pour les objets champs et les boutons de fenêtre il est possible de spécifier une fonction pour calculer l'état d'activation de l'objet. Voir Section 15.3.3 [Field object editor], page 73, et Section 15.3.10 [Window editor], page 79, pour savoir comment spécifier ce déclencheur.

La fonction déclencheur est appelée sans argument. Elle doit renvoyer NIL pour désactiver l'objet et toute autre valeur pour l'activer.

Par exemple la fonction d'activation d'un objet qui est activé lorsqu'un champ Virtuel de type '`Liste`' a un élément sélectionné ressemblerait à :

```
(DEFUN activeObjet ()
    (GETVIRTUALLISTACTIVE champ-liste-virtuel)
)
```

Voir aussi Éditeur de champ, Éditeur de fenêtre, la démo `Users.bbs`.

### 16.29.14 Calculer la description de l'enregistrement

Pour les objets table et les objets champ de type référence, une fonction déclencheur peut être définit pour calculer une description d'enregistrement (voir Section 15.3.2 [Table object editor], page 71, et Section 15.3.3 [Field object editor], page 73).

La fonction déclencheur est appelée sans aucun argument. Il doit renvoyer une seule expression ou une liste d'expressions.

Par exemple, dans une table '`Person`' avec les champs '`Name`' et '`Birthday`', la fonction pourrait ressembler à ceci :

```
(DEFUN personDescription ()
(LIST Person.Name Person.Birthday)
```

```
)
```

Voir aussi Éditeur de table, Éditeur de champ.

### 16.29.15 Déclencheur de double-clic

Pour les champs virtuels utilisant un affichage de type liste pour leurs contenus, il est possible de spécifier une fonction exécutée toutes les fois que l'utilisateur double clique sur un élément de la liste. Voir Section 15.3.3 [Field object editor], page 73, pour savoir comment spécifier ce déclencheur sur un objet champ Virtuel.

Ces déclencheurs sont appelés avec trois arguments. Le premier argument contient le numéro de ligne du champ cliqué, commençant par 1 pour la première ligne (la ligne 0 se rapporte à l'en-tête de la liste). Le deuxième argument contient le numéro de colonne en commençant par 0. Le troisième argument est un pointeur d'enregistrement auquel correspond l'élément de la liste ou NIL s'il n'a pas été produit directement à partir d'un enregistrement. Le code retour de la fonction est ignoré.

L'exemple typique de déclencheur de double-clic est le suivant :

```
(DEFUN declencheurDoubleClic (lig col enr:Table)
    ...
)
```

Ici *enr* est déclaré en tant que pointeur d'enregistrement vers la table *Table*. De cette manière il est possible d'accéder aux champs de *Table* à partir de *enr*.

Au cas où l'argument enregistrement pourrait appartenir à plusieurs tables, la construction suivante utilisant les prédicats de typage pour différencier les tables peut être utile.

```
(DEFUN doubleClickTrigger (lig col enr)
    (COND
        ((RECP Table1 enr) (SETQ Table1 enr) ...)
        ((RECP Table2 enr) (SETQ Table2 enr) ...)
        ...
    )
)
```

L'élément de la liste sur lequel l'utilisateur a cliqué peut ne se rapporter à aucun enregistrement. Dans ce cas-là, le troisième argument peut être ignoré et l'accès à l'élément de la liste se fait comme dans l'exemple suivant :

```
(DEFUN declencheurDoubleClic (lig col)
    (PRINT (NTH col (NTH lig field-virtuel)))
)
```

Voir aussi Éditeur de champ, la démo `Movie.bbs`.

### 16.29.16 Déclencheur sur glisser-déposer d'URL

Pour les champs virtuels qui utilisent le type liste pour afficher son contenu, une fonction de déclenchement peut être spécifiée qui est appelée chaque fois que l'utilisateur fait glisser-déposer d'une liste d'URL (par exemple, des noms de fichiers) sur la liste. Voir Section 15.3.3 [Éditeur d'objet champ], page 73, pour savoir comment spécifier cette fonction déclencheur pour un objet champ virtuel.

La fonction déclencheur est appelée avec un mémo qui contient toutes les URL, une par ligne. Chaque URL correspondant à un fichier local, c'est-à-dire commençant par `file://`,

est remplacée par le nom de fichier local en supprimant le préfixe `file://`. Toutes les autres URL restent inchangées. La valeur de retour de la fonction est ignorée.

Éditeur de champ.

### 16.29.17 Déclencheur de tri-déplacement

Pour les champs virtuels qui utilisent le type liste pour afficher leur contenu, une fonction déclencheur peut être spécifiée qui est appelée chaque fois que l'utilisateur fait glisser-déposer d'un élément pour trier les éléments dans la liste. Voir Section 15.3.3 [Field object editor], page 73, pour plus d'informations sur la façon de spécifier cette fonction déclencheur pour un objet champ virtuel.

Les fonctions déclencheur sont appelées avec trois arguments. Le premier argument contient l'ancien numéro de ligne de l'élément de liste, en commençant par 1 pour la première ligne (la ligne 0 fait référence à l'en-tête de la liste). Le deuxième argument contient le nouveau numéro de ligne (commençant également par 1). Le troisième argument est un pointeur vers l'enregistrement à partir duquel l'élément de liste a été généré, ou NIL si l'entrée n'a pas été générée directement à partir de celui-ci. La valeur de retour de la fonction est ignorée.

Un exemple typique d'un déclencheur tri-déplacement pour un champ *virtual-list* est le suivant.

```
(DEFUN sortDropTrigger (oldRow newRow rec)
    (MOVEREC rec (RECNUM (GETREC (NTH newRow virtual-list))))
)
```

Voir aussi Éditeur de champ, MOVEREC, RECNUM, GETREC, NTH, démo `Main.bbs`.

### 16.29.18 Calculer les entrées des listes

Pour les champs chaînes, l'objet d'IHM peut contenir une liste déroulante permettant à l'utilisateur de choisir parmi une liste de chaînes. Les entrées de cette liste peuvent être statiques ou être calculées dynamiquement par un déclencheur. Voir Section 15.3.3 [Field object editor], page 73, pour avoir des informations sur la façon de choisir entre des entrées statiques ou dynamiques et de spécifier la fonction déclencheur.

Le déclencheur pour le calcul des entrées ne requiert aucun argument. Elle doit retourner un mémo avec une entrée par ligne ou NIL pour aucune entrée.

Par exemple la fonction de calcul pourrait ressembler à ceci :

```
(DEFUN calculerEtiquettes ()
    "Tokyo\nMunich\nLos Angeles\nRome"
)
```

Voir aussi Calculer les enregistrements référencés, Éditeur de champ.

### 16.29.19 Calculer les enregistrements référencés

Pour les champs référence, l'objet graphique possède généralement un bouton permettant d'ouvrir une liste d'enregistrements parmi lesquels l'utilisateur peut faire son choix. La liste de ces enregistrements peut être calculée par un déclencheur. Voir Section 15.3.3 [Field object editor], page 73, pour avoir des informations sur la manière de spécifier le déclencheur pour les champs références. on how to specify

La fonction de calcul de la liste d'enregistrements ne requiert aucun argument. Elle doit retourner une liste devant contenir des enregistrements de la table référencée. Tout

enregistrement de cette table est ajouté à la liste affichée. Les éléments qui ne sont pas des enregistrements de la table référencée sont ignorés silencieusement.

L'exemple suivant illustre une fonction typique de calcul d'enregistrements référencés. Disons qu'un projet contient une table'`Persone`' avec un champ booléen '`Femme`'. Alors la fonction de calcul suivante n'affiche que les femmes dans la liste déroulante :

```
(DEFUN calculerEnregistrementsFemme ()
    (SELECT Personen FROM Personne WHERE Femme)
)
```

Voir aussi Calculer les entrées des listes, Éditeur de champ.

## 16.30 Liste des fonctions obsolètes

Les fonctions suivantes sont obsolètes.

- `GETDISABLED`
- `SETDISABLED`
- `GETWINDOWDISABLED`
- `SETWINDOWDISABLED`

Les fonctions obsolètes ne fonctionnent plus comme attendu et leur appel est soit ignoré (donnant une non-opération), soit ouvre une fenêtre d'avertissement, soit cause une erreur selon le réglage du menu '`Programme - Fonctions obsolètes`' (voir Section 7.2.10 [Obsolete functions], page 39).

Il est recommandé d'enlever ces fonctions des programmes et d'implémenter la fonctionnalité en utilisant le réglage activé/désactivé des objets champ et des boutons de fenêtre (voir Section 16.29.13 [Compute enabled function], page 163).

# 17 Interface ARexx

L'interface ARexx n'existe que pour la version Amiga de BeeBase.

ARexx est une interface standard pour des programmes Amiga qui permet d'accéder aux fonctions et données d'autres programmes. BeeBase fournit un port ARexx avec un nombre de commandes restreint mais néanmoins suffisant pour permettre à un programme externe d'effectuer des opérations comme si c'était BeeBase lui-même qui les effectuaient. De plus l'interface Arexx de BeeBase possède un mécanisme de transaction similaire à d'autres bases de données relationnelles.

Des exemples de scripts Arexx pour BeeBase sont disponibles dans '`rexx`'.

## 17.1 Nom du port

Le port ARexx dans BeeBase est nommé '`BeeBase.`*n*' où *n* est un compteur qui démarre à 1. Habituellement, si vous lancez une seule fois BeeBase, le nom du port sera '`BeeBase.1`'.

Vous devez déclarer le nom du port ARexx via `address` avant de lancer l'une des commandes ARexx de BeeBase. Le bout de programme suivant montre comment vérifier la présence du port ARexx dans BeeBase, comment lancer BeeBase le cas échéant, ainsi que comment interagir avec le port.

```
if ~show(ports, BeeBase.1) then
do
    address command 'run <nil: >nil: BeeBase:BeeBase -n'
    address command 'waitforport BeeBase.1'
end

address BeeBase.1
```

Consultez également l'exemple de script ARexx `address.rexx`.

## 17.2 Syntaxe des commandes

Après avoir contacté le port ARexx de BeeBase, vous pourrez lancer n'importe quelle commande ARexx de BeeBase. La syntaxe est similaire à celle d'autres implémentations :

*cmd* [*arg1* ...]

où *cmd* est une des commandes décrites plus loin bas dans ce chapitre, et *arg1* ... sont des arguments optionnels de cette commande.

Puisque l'interpréteur ARexx évalue la ligne de commande avant de l'envoyer à BeeBase, il peut être utile parfois de mettre entre guillemets certains, voire tous les arguments. Il est recommandé d'utiliser des guillemets simples (') pour les arguments qui ne seront plus utilisés ultérieurement par l'interpréteur ARexx. Ainsi vous pouvez toujours utiliser des guillemets doubles (") pour des arguments, par exemple pour des constantes de texte. De plus, vous pouvez intégrer la valeur des variables ARexx en leur enlevant leurs guillemets. Voici un exemple utilisant la commande BeeBase `eval` :

```
recherche = ''
eval handle 'select Nom from Personne where (like Nom "*'recherche'*")'
```

Consultez également Eval.

## 17.3 Codes retour

Après avoir lancé l'une des commandes ARexx de BeeBase, plusieurs variables ARexx sont mises à jour avec le résultat de la commande. Pour afficher tous les résultats d'une commande, vous devez activer l'option de résultats ARexx en ajoutant les lignes suivantes au début de votre script ARexx :

```
options results
```

Il existe 3 variables ARexx qui peuvent être définies par l'interface ARexx de BeeBase : *rc*, *results* et *lasterror*. La variable *rc* est toujours définie et indique le succès ou l'échec d'une commande. Si une commande est lancée avec succès, la variable *results* est initialisée avec le résultat de la commande alors que dans le cas d'un échec de la commande, la variable *lasterror* peut contenir des informations supplémentaires décrivant l'erreur.

Pour la variable *rc* il existe les codes de retour suivants :

| Code retour | Signification |
|---|---|
| 0 | Succès. La variable *result* contient le résultat actuel. |
| -1 | Erreur d'implémentation. Ne devrait jamais se produire. |
| -2 | Mémoire insuffisante. |
| -3 | Commande ARexx inconnue. |
| -4 | Erreur syntaxique. |
| -10 | Autre erreur. Une description de l'erreur est donnée par *lasterror*. |
| -11 | Erreur interne. Ne devrait jamais se produire. |
| -12 | Erreur de compilation (uniquement pour la commande compile). |

Dans le cas où *rc* <= -10, la variable *lasterror* contient une description valide de l'erreur. Des codes d'erreur supplémentaires pourraient être ajoutés dans le futur de façon à avoir un rapport d'erreur plus détaillé.

Voici un bout de code typique qui montre comment évaluer le résultat d'une commande ARexx de BeeBase :

```
eval handle 'select * from Comptes'
if (rc == 0) then
    say result
else if (rc == -1) then
    say "Erreur d'implémentation"
else if (rc == -2) then
    say "Mémoire insuffisante"
else if (rc == -3) then
    say "Commande inconnue"
else if (rc == -4) then
    say "Erreur syntaxique"
else if (rc <= -10) then
    say lasterror
else
    say "Erreur : " rc
```

## 17.4 Quit

La commande `quit` provoque la fermeture du programme BeeBase. Veuillez également consulter la documentation de MUI.

## 17.5 Hide

La commande `hide` icônifie toutes les fenêtres ouvertes de BeeBase. Veuillez également consulter la documentation de MUI.

## 17.6 Show

La commande `show` désicônifie BeeBase et rouvre les fenêtres. Veuillez également consulter la documentation de MUI.

## 17.7 Info

La commande `info` retourne les renseignements suivants à propos de l'application MUI : titre, concepteur, droits d'auteur, description, version, port et écran.

| **Commande** | **Valeur de** *result* |
|---|---|
| `info title` | `Titre de l'application` |
| `info author` | `Concepteur de l'application` |
| `info copyright` | `Droits d'auteur` |
| `info description` | `Description` |
| `info version` | `Numéro de version` |
| `info base` | `Nom du port ARexx` |
| `info screen` | `Nom de l'écran public` |

Veuillez également consulter la documentation de MUI.

## 17.8 Help

La commande `help` écrit dans un fichier toutes les commandes ARexx disponibles de l'application MUI.

`help` *`nom-du-fichier`*

Les commandes ARexx sont affichées en respectant la syntaxe de l'interpréteur de commandes AmigaDOS. Veuillez également consulter la documentation de MUI ainsi que le manuel d'AmigaDOS pour la syntaxe de la ligne de commande.

## 17.9 Compile

La commande `compile` compile le code source d'un programme externe.

`compile` *`source`* `[update]`

Cette commande compile le fichier source d'un programme externe au projet dont le nom de fichier est spécifié avec la commande *source*. En cas de succès, la commande retourne

la valeur 0, et si `update` est spécifié, le fichier source externe est réécrit. La mise à jour du fichier source permet la mise en évidence des mots clefs BeeBase. Un programme compilé avec succès est considéré comme le programme du projet et est utilisé lors de l'exécution de déclencheurs.

Dans le cas d'un échec lors de la compilation, le code d'erreur -12 est retourné et *lasterror* est défini sur 4 lignes de texte :

- La première ligne contient le nom du fichier qui a généré l'erreur.
- La deuxième ligne contient le numéro de la ligne où l'erreur s'est produite en commençant à 1.
- La troisième ligne contient le numéro de la colonne où l'erreur s'est produite en commençant à 1.
- La quatrième ligne décrit l'erreur sous forme de texte compréhensible.

Un projet doit absolument être déjà ouvert avant de lancer la commande `compile` pour compiler son code source externe. Si aucun projet externalisant son code source dans le fichier spécifié par *source* n'est trouvé, un code d'erreur <= -10 (mais différent de -12) est retourné et *lasterror* est défini.

## 17.10 Connect

La commande `connect` ouvre la communication vers un projet BeeBase.

`connect` *`nom-du-projet`* `[GUI]`

La commande vérifie en premier lieu si le projet défini par `nom-du-projet` est déjà ouvert et l'ouvre le cas échéant. Un projet n'est lancé qu'une seule fois et les diverses connexions vers le même projet partagent l'accès à la base de données. Ensuite un handle de communication (ou identifiant unique) est généré. Un handle de communication est une valeur entière non nulle. Si le mot-clef `GUI` fait partie de la ligne de commande, alors l'interface graphique du projet BeeBase est également lancée. Sinon aucune interface graphique n'apparaît ce qui permet d'exécuter des commandes ARexx de BeeBase en tâche de fond sans l'intervention directe de l'utilisateur.

En cas de succès, la valeur 0 est retournée et *result* prend la valeur du handle.

Exemple : '`connect "BeeBase:Demos/Movie.bbs"`' établit une connexion vers la base de données d'exemple de films.

Consultez également Disconnect, Connections, Codes retour.

## 17.11 Disconnect

La commande `disconnect` termine une communication existante.

`disconnect` *`handle`*

Termine la connexion à la base de donnée désignée par *handle*. Si c'est la seule connexion vers le projet référencé par *handle* et si le projet ne dispose d'aucune interface graphique, alors le projet peut être fermé et déchargé de la mémoire. Sinon le projet reste ouvert.

Exemple : '`disconnect 1`' termine la connexion ayant le handle 1.

Consultez également Connect, Connections, Codes retour.

## 17.12 connections

La commande `connections` donne des renseignements sur les connexions existantes.

```
connections
```

En cas de succès, `connections` retourne la valeur 0 et définit *result* sous forme de texte compréhensible dont chaque ligne est une connexion avec une valeur de handle et un nom de projet.

Exemple : la variable *result* après un appel à `connections` pourrait se présenter sous la forme suivante :

```
3 BeeBase:Demos/Accounts.bbs
5 BeeBase:Demos/Movie.bbs
6 BeeBase:Demos/Movie.bbs
7 BeeBase:Demos/Movie.bbs
```

Consultez également Connect, Disconnect, Codes retour.

## 17.13 Eval

L'interface principale du port Arexx de BeeBase permettant la récupération et la mise à jour des données est gérée par la commande `eval`.

```
eval handle commande-lisp
```

La commande `eval` exécute la commande spécifiée par *commande-lisp* (écrite dans le langage lisp de BeeBase) sur le projet correspondant à *handle*. Un handle peut être obtenu via la commande `connect`. La commande *commande-lisp* peut être n'importe quelle expression du langage de programmation de BeeBase. La parenthèse la plus externe d'une expression est optionnelle et peut éventuellement être omise. Il est recommandé d'entourer *commande-lisp* de guillemets simples comme décrit dans Section 17.2 [Syntaxe des commandes], page 167.

En cas de succès, `eval` retourne la valeur 0 et définit *result* comme une chaîne de caractères de la valeur de retour de *commande-lisp*. La chaîne de caractères est faite de façon à avoir une idée du type de données retourné, par exemple un texte est entouré par des guillemets doubles alors qu'une liste est entourée de parenthèses dont les éléments sont séparés par des espaces ou des caractères retour chariot. Si vous voulez avoir un format spécifique, utilisez votre propre format dans la commande lisp spécifiée.

Si vous modifiez la base de données via la commande `eval` sans avoir auparavant lancé une transaction (voir Section 17.14 [Transaction], page 172) les changements deviennent automatiquement permanents (validation automatique). À l'inverse si vous avez lancé une transaction avant d'appeler `eval` les changements seront conservés en mémoire jusqu'à ce que la commande `commit` les rende permanents ou que la commande `rollback` les annule.

Exemple:

```
options results
address BeeBase.1
connect "BeeBase:Demos/Movie.bbs"
if rc = 0 then
do
    handle = result
```

```
    eval handle 'select Titre, Réalisateur from Films'
end
if rc = 0 then
    say result
```

Le résultat de l'exemple ci-dessus pourrait ressembler à ceci :

```
( ( "Titre" "Réalisateur" )
  ( "Batman" "Tim Burton" )
  ( "Batman Returns" "Tim Burton" )
  ( "Chérie, vote pour moi" "Ron Underwood" )
  ( "Tequila Sunrise" "Robert Towne" )
  ( "Mad Max" "George Miller (II)" )
  ( "Braveheart" "Mel Gibson" )
  ( "2010 - L'année du premier contact (L'odyssée continue)" "Peter Hyams" ) )
```

Consultez également Connect, Syntaxe des commandes, Codes retour, Transaction, Commit, l'exemple de script ARexx `movies.rexx`.

## 17.14 Transaction

Le port ARexx de BeeBase permet d'effectuer des transactions dans la base de données. Une transaction consiste en un ensemble d'opérations appliquées sur la base de données permettant la modification des données. Une transaction peut être soit exécutée et rendue permanente définitivement (`commit`), soit au contraire annulée à n'importe quel point de la transaction (`rollback`). On lance une transaction avec la commande suivante :

`transaction` *handle*

où *handle* fait référence à un projet obtenu par la commande `connect` (voir Section 17.10 [Connect], page 170).

Après lancement de la commande `transaction` vous pouvez accumuler autant de commandes `eval` que vous le désirez sans affecter la base de données. Pourtant il va vous falloir décider à un moment si vous voulez rendre les modifications permanentes (voir Section 17.15 [Commit], page 172) ou revenir en arrière, avant le lancement de la commande `transaction` (voir Section 17.16 [Rollback], page 173).

Après lancement de la commande `transaction` l'accès au projet correspondant est exclusif au handle spécifié. Ainsi, les autres programmes voulant accéder à la base de données (y compris l'utilisateur en passant par l'interface graphique) sont bloqués (ou différés dans le cas d'une autre connexion ARexx) jusqu'à ce que l'accès exclusif soit levé, en exécutant la commande `commit` ou `rollback`.

Habituellement la commande `transaction` retourne la valeur 0. Si une autre connexion ARexx requiert l'accès exclusif au même projet spécifié par *handle*, la demande est bloquée jusqu'à ce que l'autre connexion se termine soit en validant, soit en annulant la transaction.

Consultez également Eval, Commit, Rollback, Codes retour.

## 17.15 Commit

La commande `commit` s'utilise à la fin d'une transaction pour valider les changements de manière définitive.

`commit` *handle*

La commande `commit` met fin à une transaction (voir Section 17.14 [Transaction], page 172) en enregistrant le projet auquel fait référence *handle*. En cas de succès, la commande `commit` retourne la valeur 0. Si aucune transaction n'a été lancée avant l'appel de la commande `commit` ou s'il se produit une erreur, une valeur différente de 0 est retournée.

Consultez également Rollback, Transaction, Codes retour.

## 17.16 Rollback

Pour annuler les changements d'une transaction, utilisez la commande `rollback`.

```
rollback handle
```

Tous les changements effectués dans un projet référencé par *handle* depuis le début de la transaction en cours (voir Section 17.14 [Transaction], page 172) sont annulés et le projet revient à l'état où il se trouvait avant le lancement de la transaction. En cas de succès, la commande `rollback` retourne la valeur 0.

Consultez également Commit, Transaction, Codes retour.

# Menus

## Menu Projet

**Information**
: Informations sur le projet, voir Section 6.2 [Info], page 31.

**Nouveau**
: Débute un nouveau projet, voir Section 6.3 [New project], page 31.

**Nettoyer - Projet**
: Réinitialise le projet, voir Section 6.4 [Clear project], page 31.

**Nettoyer - Enregistrements**
: Efface tous les enregistrements, voir Section 6.4 [Clear project], page 31.

**Ouvrir - Projet**
: Charge un projet, voir Section 6.5 [Open project], page 31.

**Ouvrir - Démo**
: Charge un projet démo, voir Section 6.5 [Open project], page 31.

**Recharger**
: Recharger une nouvelle version du projet, voir Section 6.5 [Open project], page 31.

**Enregistrer**
: Sauve le projet sur le disque, voir Section 6.6 [Save project], page 32.

**Enregistrer & Réorganiser**
: Sauve et réorganise le projet, voir Section 6.6 [Save project], page 32.

**Enregistrer & Réorganiser en**
: Sauve et réorganise avec un nouveau nom, voir Section 6.6 [Save project], page 32.

**Passer en mode Administrateur**
: Restreindre l'accès à la structure du projet, voir Section 6.7 [Admin and user mode], page 32.

**Modifier le mot de passe Administrateur**
: Identification pour le mode administrateur, voir Section 6.7 [Admin and user mode], page 32.

**Décharger les enregistrements**
: Décharge tous les enregistrements sur disque, voir Section 6.8 [Swap records], page 33.

**Éditeur de structure**
: Ouvre l'éditeur de structure, voir Chapitre 15 [Structure editor], page 65.

**Exporter la structure**
: Résumé de toutes les tables et champs, voir Section 15.4 [Export structure], page 80.

**Fermer**
: Lorsque le travail sur un projet est terminé, voir Section 6.9 [Close project], page 33.

**Enregistrer et Fermer**
Sauve puis ferme le projet, voir Section 6.9 [Close project], page 33.

**Quitter** Sortie de BeeBase, voir Chapitre 3 [Getting started], page 6.

## Menu Préférences

**Formats** Format des réels et des décimaux, voir Section 7.1.1 [Formats], page 34.

**Éditeur externe**
Configure l'éditeur de texte externe, voir Section 7.1.2 [External editor], page 34.

**Visionneuse externe**
Configure l'outil utilisé pour visionner les fichiers, voir Section 7.1.3 [External viewer], page 35.

**Boutons dans cycle de tabulation**
Ajoute les boutons dans le cycle de tabulation, voir Section 7.1.4 [Extra buttons in Tab chain], page 35.

**Champ suivant via <Entrée>**
Se déplace au champ suivant lors de la pression de la touche *`Entrée`*, voir Section 7.1.5 [Advance on Enter], page 36.

**Confirmer la sortie**
Requête de sécurité lors de la sortie de BeeBase, voir Section 7.1.6 [Confirm quit], page 36.

**MUI** Préférences de MUI, voir Section 7.1.7 [MUI], page 36.

**Cache d'enregistrements**
Taille du cache d'enregistrements, voir Section 7.2.1 [Record memory], page 36.

**Confirmer la suppression**
Requête de sécurité lors de la suppression des enregistrements, voir Section 7.2.2 [Confirm delete record], page 37.

**Chemins relatifs au projet**
Comment les chemins relatifs sont traités, voir Section 7.2.3 [Paths relative to project], page 37.

**Confirmer enregistrement et réorganisation**
Requête de sécurité pour la sauvegarde et la réorganisation d'un projet, voir Section 7.2.5 [Confirm save & reorg], page 38.

**Enregistrer comme défaut**
Enregistre les réglages du projet pour les projets futurs, voir Section 7.3 [Save as default], page 40.

## Menu Journal

**Activer le journal ?**
Comment activer ou désactiver le journal, voir Section 8.2 [Activate logging], page 41.

**Mode de journalisation**
: Destination des entrées du journal, voir Section 8.3 [Logging mode], page 42.

**Définir le fichier journal**
: Définir un fichier journal externe, voir Section 8.4 [Set log file], page 42.

**Définir le tag du journal**
: Définir un tag pour les nouvelles entrées du journal, voir Section 8.5 [Set log label], page 42.

**Importer un journal**
: Importation d'entrées de journal depuis un fichier, voir Section 8.6 [Import log], page 42.

**Exporter le journal**
: Exportation d'entrées de journal vers un fichier, voir Section 8.7 [Export log], page 43.

**Vider le journal**
: Suppression de toutes les entrées de journal, voir Section 8.8 [Clear log], page 43.

**Appliquer les modifications**
: Appliquer les modifications d'un journal au projet, voir Section 8.9 [Apply changes], page 43.

**Afficher le journal**
: Comment afficher toutes les entrées de journal, voir Section 8.10 [View log], page 43.

## Menu Table

**Nouvel enregistrement**
: Ajoute un nouvel enregistrement, voir Section 9.2 [Adding records], page 45.

**Dupliquer l'enregistrement**
: Recopie un enregistrement, voir Section 9.2 [Adding records], page 45.

**Supprimer l'enregistrement**
: Lorsqu'un enregistrement n'est plus nécessaire, voir Section 9.4 [Suppression d'enregistrement], page 48.

**Supprimer tous les enregistrements**
: Pour repartir de zéro, voir Section 9.4 [Deleting records], page 48.

**Atteindre l'enregistrement**
: Parcourt les enregistrements, voir Section 9.5 [Browsing records], page 48.

**Changer le filtre**
: Spécifie une expression de filtrage, voir Section 10.1.2 [Changing filters], page 49.

**Changer l'ordre**
: Comment spécifier un ordre de tri, voir Section 11.4 [Changing orders], page 52.

**Réordonner tous les enregistrements**
: Lorsque les enregistrements ne sont plus triés, voir Section 11.5 [Reorder all records], page 53.

**Rechercher**
: Comment rechercher un enregistrement, voir Section 12.1 [Search dialog], page 54.

**Rechercher suivant**
: Aller à l'enregistrement correspondant suivant, voir Section 12.2 [Forward/backward search], page 54.

**Rechercher précédent**
: Aller à l'enregistrement correspondant précédent, voir Section 12.2 [Forward/backward search], page 54.

**Importer des enregistrements**
: Comment importer des enregistrements, voir Section 13.3 [Importing records], page 57.

**Exporter des enregistrements**
: Comment exporter des enregistrements, voir Section 13.4 [Exporting records], page 57.

**Afficher tous les enregistrements**
: Affiche tous les enregistrements d'une table, voir Section 9.6 [View all records], page 48.

## Menu Programme

**Éditer** Où saisir un programme BeeBase, voir Section 16.1 [Program editor], page 81.

**Compiler** Compilation d'un programme, voir Section 16.1 [Program editor], page 81.

**Code source du programme**
: Code source interne ou externe, voir Section 7.2.7 [Program source], page 38.

**Nettoyer les sources des programmes externes**
: Efface les fichiers sources externes lorsqu'ils ne sont plus nécessaires, voir Section 7.2.8 [Cleanup external program source], page 38.

**Inclure les informations de débogage**
: Compile avec ou sans les informations de débogage, voir Section 7.2.9 [Program debug information], page 38.

**Fonctions obsolètes**
: Comment gérer les appels aux fonctions obsolètes, voir Section 7.2.10 [Obsolete functions], page 39.

**Trier les déclencheurs**
: Tri alphabétique dans les fenêtres flottantes, voir Section 7.2.11 [Sort trigger functions], page 39.

**Répertoire d'inclusion**
: Où rechercher les fichiers d'inclusion externes, voir Section 7.2.12 [Program include directory], page 39.

**Fichier de sortie**
: Où vont les sorties du programme, voir Section 7.2.13 [Program output file], page 39.

**Requêtes** Ouvrer l'éditeur de requêtes, voir Section 14.2 [Query editor], page 59.

## Menu Aide

**Contenu** Ce manuel utilisateur.

**À propos de**
: À propos de BeeBase, voir Chapitre 1 [Copying], page 1.

**À propos de MUI**
: À propos de Magic User Interface, voir Section 1.5 [Third party material], page 2.

# Remerciements

Merci à :

- Ralph Reuchlein (Ralphie) pour les rapports de bogue, les idées et suggestions, ainsi que pour la traduction allemande du manuel de BeeBase.

  Ralphie a également créé la site web initial de BeeBase `https://beebase.sourceforge.io`.
- Pascal Marcelin pour sa foi en BeeBase et l'Amiga, et pour le premier serveur web se connectant à BeeBase via son port ARexx.
- Christoph Pölzl et Sébastien Pölzl pour les graphismes utilisés dans BeeBase, pour le jeu d'icônes PNG et pour les tests sous MorphOS.
- Alexandre Balaban pour la traduction française du manuel utilisateur (beaucoup aidé par Lionel Muller et Gilles Mathevet) et pour le portage de BeeBase sur AmigaOS 4.
- Stéphane Aulery pour continuer la traduction française, pour avoir suggéré de renommer MUIBase en BeeBase, et pour avoir donner au site web de BeeBase une mise à jour majeur.
- Harold Kinds pour ces nombreux projets de démonstration.
- Ilkka Lehtoranta pour avoir terminé le portage MorphOS.
- Thomas Fricke et Adrian Maleska pour les divers graphismes améliorant l'apparence de BeeBase.
- Martin Merz pour les icônes Mason.
- Mats Granstrom pour les bêta-tests de BeeBase et pour l'écriture du tutoriel.
- John Hertig, Harold Kinds, Magnus Hammarstrom, Henning Thilemann, Joseph Durchalet, André Schenk, Klaus Gessner et Oliver Roberts pour les idées et les bêta-tests.
- A tous les traducteurs qui rendent accessibles BeeBase dans leur langue, nombreux sont membres du Translation Project `https://translationproject.org` : Jaap Verhage (Hollandais), Stéphane Aulery (Français), Samir Hawamdeh (Italien), Sharuzzaman Ahmat Raslan (Malais), Miroslav Nikolic (Serbe), Benno Schulenberg (Espagnol), et Yuri Chornoivan (Ukrainien).
- Jernej Simoncic pour l'autorisation d'utiliser son script d'installation Windows de The Gimp comme base pour celui de BeeBase pour Windows.

# Auteur

BeeBase est développé par :

Steffen Gutmann
Email: beebase.bbs@gmail.com

# Index des fonctions

## E



## F



## G



## H



## I



## L



## M

## N



## O



## P



## R



## S



## T



## U



## V



## W

# Y

# Index des concepts

## F



## G



## I



## J



## L



## M



## N

## O



## P



## Q



## R



## S



## T



## V