Manuel de PureBasic
5.60

http://www.purebasic.com/

3 mars 2017

Table des matières

Première partie

Thèmes généraux

Chapitre 1

Introduction

PureBasic est un langage de programmation de "haut niveau" basé sur les règles du langage BASIC. Il est identique à tout autre compilateur BASIC que vous auriez pu déjà utiliser. PureBasic a été créé pour être aussi accessible au débutant qu'à l'expert, il est donc très facile à apprendre. La compilation est très rapide, presque instantanée. Nous avons consacré beaucoup de temps et d'efforts pour vous proposer un langage rapide, fiable et convivial.
La syntaxe de PureBasic est simple, mais ses possibilités sont infinies grâce à certaines caractéristiques évoluées comme, entre autres, les pointeurs, structures, procédures, listes dynamiques, etc. Le programmeur expérimenté n'aura aucune difficulté à accéder aux structures du système d'exploitation et aux API's.
PureBasic est un langage portable qui fonctionne actuellement sur les ordinateurs PC dotés du système d'exploitation Windows, Linux ou Mac OS X. Cela signifie qu'un même programme peut être compilé sur chacune de ces machines et en exploiter toute la puissance. PureBasic est un compilateur. Il n'utilise donc pas de code intermédiaire ou de machine virtuelle, mais produit un code optimisé directement exécutable par la machine ou le système d'exploitation sur lequel il est compilé. Les bibliothèques externes sont écrites et optimisées manuellement en assembleur. Les commandes sont donc très rapides, souvent plus rapides que leurs équivalentes écrites en langage C/C++.
Pour information, les passionnés de machines anciennes trouveront une version en opensource pour AmigaOS (680x0 et PowerPC) ici

Caractéristiques techniques

- Supporte nativement les microprocesseurs x86 et x64
- Définitions intégrées de tableaux, listes dynamiques, structures complexes, pointeurs et variables
- Types supportés : Byte (8-bit), Word (16-bit), Long (32-bit), Quad (64-bit), Flottant (32 et 64-bit) et Caractère (8 ou 16-bit)
- Types définis par l'utilisateur (structures)
- Définition intégrée des chaines de caractères avec de nombreuses fonctions dédiées, y compris en mode unicode
- Puissant support des macros
- Support des constantes et valeurs octales et hexadécimales
- Réducteur d'expression (regroupement des constantes et valeurs explicites)
- Calcul arithmétique standard avec respect de la priorité des opérateurs et parenthèses : +, -, /, *, and, or, lsl, asl, lsr, asr
- Compilation extrêmement rapide
- Support des procédures pour une programmation structurée avec variables globales et locales
- Supporte tous les mots-clé évolués du langage Basic : If-Else-EndIf, Repeat-Until, For-Next, etc.
- Support de bibliothèques externes pour la manipulation d'objets Windows : Images, fenêtres, composants graphiques, DirectX, etc.
- Les bibliothèques externes sont également toutes écrites en langage assembleur optimisé pour plus de rapidité et de compacité

- Les API's de Windows (Linux et OSX) sont supportées et considérées comme des mots-clé du BASIC
- Langage assembleur intégré permettant d'insérer toute commande ou routine assembleur dans le corps du programme Basic
- Constantes et structures précompilées pour une compilation encore plus rapide
- Débogueur intégré pour suivre l'exécution d'un programme et corriger les erreurs plus facilement
- Options de compilation configurables
- Editeur dédié avec syntaxe automatiquement surlignée
- Système convivial, facile à installer, à comprendre et utiliser
- Entièrement en français

Chapitre 2

Licence et Conditions

Ce programme est proposé sans aucunes garanties. Fantaisie Software ne peut en aucun cas être tenu responsable des dégâts occasionnés par PureBasic. Vous utilisez ce logiciel à vos risques et périls.
La version de démonstration de PureBasic peut être distribuée librement tant que le contenu de l'archive reste intacte. Il n'est permis de modifier, changer l'organisation de l'archive sans la permission écrite de Fantaisie Software.
La version complète de PureBasic ne doit en aucun cas être cédée à une personne autre que l'acheteur du logiciel.
Fantaisie Software détient tous les droits sur PureBasic et ses composants. Aucun module ne peut être utilisé dans une autre application sans l'autorisation de Fantaisie Software. Il est interdit d'encapsuler ou d'utiliser directement les commandes de haut niveau de PureBasic dans un autre langage de programmation que ce soit sous forme de bibliothèques dynamiques (DLL) ou statiques. Cette règle ne s'applique pas pour les utilisateurs possédant une licence de PureBasic.
PureBasic utilise le moteur 3D OpenSource OGRE et sa licence peut être consultée ici .
Les sources modifiées spécialement pour PureBasic du moteur 3D OGRE peuvent être téléchargées à l'adresse suivante : http ://www.purebasic.com/OgreSources.zip.
PureBasic utilise la bibliothèque OnError, écrite et (c) 2003 par Siegfried Rings et Sebastian Lackner.
PureBasic utilise le composant d'édition Scintilla et sa licence peut être consultée ici .
La bibliothèque XML utilise le parseur XML expat et sa licence peut être consultée ici .
La bibliothèque RegularExpression utilise PCRE. et sa license peut être consultée ici .

Chapitre 3

Configuration système requise

PureBasic fonctionne sur les versions 32 bits et 64 bits de Windows XP, Vista, Windows 7, Windows 8 et Windows 10, Linux (noyau 2.2 et ultérieur) et MacOS X (10.6 et ultérieur).
Si vous rencontrez un problème sur l'une des configurations ci-dessus, n'hésitez pas à nous en informer.

Windows

Afin d'utiliser les nombreuses fonctions graphiques 3D ainsi que les Sprites , il est nécessaire d'utiliser la version 9.0c de DirectX. Windows Vista et les versions ultérieures de Windows contiennent déjà cette version de DirectX, mais pour les versions antérieures de Windows, il faudra procéder à une installation manuelle.
Vous pouvez la télécharger et l'installer via le "DirectX End-User Runtime Web Installer" ici :
http ://www.microsoft.com/en-us/download/details.aspx ?displaylang=en&id=35.
En français : https ://www.microsoft.com/fr-fr/download/details.aspx ?id=20265

Linux

Paquets nécessaires pour utiliser PureBasic sous Linux :

- sdl 1.2 devel (pour les commandes multimédias)
- gtk 2+ (Pour les programmes GUI)
- libstdc++ devel
- gcc correctement installé
- iodbc et iodbc-devel doivent être installés pour utiliser les commandes Database (voir www.iodbc.org)
- libwebkit doit être installé pour utiliser WebGadget() .
- xine et xine-devel pour les commandes Movie Pour plus d'information, consulter les fichiers INSTALL et README.

MacOS X

Les outils de développement Apple (Apple Developer Tools) doivent être installés avant de pouvoir utiliser PureBasic. Ils sont fournis sur les CDs (ou DVD) d'installation de MacOS X ou disponible sur le site web d'Apple : http ://developer.apple.com/
Pensez à utiliser la dernière version des différents outils (par exemple XCode), adaptée à la bonne version de votre OS ! Les outils de ligne de commandes doivent être installés depuis XCode.

Chapitre 4

Installation

Pour installer PureBasic, cliquez sur l'icône d'installation et suivez les instructions. Cliquez ensuite sur l'icône PureBasic qui se trouve dans le menu démarrer ou sur le bureau.
Pour utiliser le compilateur en ligne de commande, il est conseillé de rajouter le répertoire "compilers\" de l'installation PureBasic à la variable d'environnement 'PATH'. Ainsi la commande 'pbcompiler' sera accessible à partir de n'importe quelle console.
Note : Pour éviter les conflits avec une ancienne version, installez toujours une nouvelle version de PureBasic dans un nouveau répertoire .

Chapitre 5

Obtenir PureBasic

PureBasic est un langage de programmation très performant à un prix raisonnable. En achetant PureBasic, vous encouragez son développement et vous participez à son évolution future. Les mises à jour sont gratuites et illimitées, ce qui implique que vous ne paierez PureBasic qu'une seule et unique fois, contrairement à de nombreux autres logiciels. Encore plus fort, vous avez droit à toutes les versions de PureBasic pour toutes les plateformes actuellement supportées ,Windows, Linux et MacOS (AmigaOS pour les versions antérieures). Merci encore pour votre confiance et votre support !

La version de démonstration est limitée de la manière suivante :

- Impossible de créer une DLL
- Impossible d'utiliser les fonctions externe (API de l'OS)
- Pas de kit de développement pour ajouter des libraries à PureBasic
- Nombre maximum de lignes pour un programme : 800

Prix : La version complète de PureBasic :

Prix pour la version complète : 79 Euros
Prix spéciaux pour les licences entreprises (499 euros) et licences éducations (199 euros pour une classe).

Paiement en ligne sécurisé

http ://www.purebasic.com
La version complète sera envoyée par e-mail (email contenant un lien à télécharger) dès réception de la commande.

Chapitre 6

Contacts

Veuillez envoyer vos suggestions, rapports de bugs, ou si vous voulez simplement nous contacter à une des adresses suivantes :

Frédéric 'AlphaSND' Laboureur

Fred 'AlphaSND' est le fondateur de Fantaisie Software et le programmeur principal de PureBasic. Toutes les suggestions, rapports de bugs etc... lui seront adressés à :

Courrier :

Fantaisie Software
10, rue de Lausanne
67640 Fegersheim
France

e-mail : fred@purebasic.com

André Beer

André se charge de la traduction et du support allemand de PureBasic (manuel et site web).
e-mail : andre@purebasic.com

Chapitre 7

Remerciements

Nous voulons remercier les nombreuses personnes qui nous ont aidés dans cet ambitieux projet. Il n'aurait pas pu prendre vie sans eux !

- Tous les utilisateurs enregistrés : Pour supporter activement le développement de ce logiciel... Encore merci !

Codeurs

- **Roger Beausoleil** : Le premier à croire vraiment en ce projet, et son aide précieuse dans le design initial de PureBasic.
- **Andre Beer** :Pour prendre le temps de traduire chaque nouvelle version en Allemand...
- **Francis G.Loch** : Pour avoir corrigé toutes les fautes d'orthographe dans le manuel anglais. Encore merci !
- **Steffen Haeuser** : Pour avoir donné de son temps lors de la création de la version Amiga PowerPC (PPC) de PureBasic.
- **Martin Konrad** : Un 'bug reporter' fantastique qui a mis à nus la plupart des bugs présents dans PureBasic Amiga.
- **James L. Boyd** : Pour avoir trouvé beaucoup de bugs dans la version Windows x86 et pour nous avoir donné une tonne de boulot supplémentaire :).
- **Les** : Pour avoir corrigé la version anglaise du site web et du guide de référence de PureBasic.
- **L'équipe NAsm** : Pour avoir créé un assembleur étonnant utilisé pour le développement des bibliothèques x86
- **Tomasz Grysztar** : Pour FAsm, un excellent assembleur x86 actuellement utilisé par PureBasic. http ://flatassembler.net/
- **Pelle Orinius** : Pour ses excellents outils (linker et compilateur de ressources de PellesC) qui sont actuellement utilisés par PureBasic.
- **Jacob Navia** :Pour son linker (Lcc32) qui était utilisé dans les versions antérieures de PureBasic
- **Frank Wille** : Pour nous avoir permis d'inclure ses deux assembleurs (PhxASS et pasm) dans les versions 680x0 et PowerPC de PureBasic. Enfin pour son aide en ce qui concerne la programmation assembleur PPC.
- **Danilo**, **Rings**, **Paul**, **Franco** and **MrVain/Secretly** : Pour les rapports de bugs et les suggestions interminables qui ont permis de stabiliser et d'améliorer PureBasic x86 d'une manière très rapide.
- **Sylvain B.** : Pour avoir traduit la première partie de la documentation française.
- **Francois Weil** : Pour avoir traduit le reste (une grande majorité) de la documentation française, et ce en moins d'une semaine ! Encore merci...
- **David 'Tinman' McMinn** et **Timo 'Freak' Harter** : Pour améliorer considérablement l'aide du PureBasic et ainsi sa découverte au plus grand nombre !
- **Danilo** : Pour avoir fait un travail énorme sur l'éditeur et même sur le jeu interne des commandes, sans

oublier ses excellentes suggestions quant à l'optimisation et la réduction des exécutables... Encore merci !
- **Marc Vitry** : Pour avoir partagé ses sources de sa bibliothèque 'SerialPort', qui ont servi de base à celle incluse dans le PureBasic
- **Stefan Moebius** : Pour avoir envoyé les sources de son sous-système DX9, qui ont servi de base au sous-système DX actuel
- **Comtois**, **G-Rom** et **T-Myke** : pour avoir amélioré drastiquement la partie 3D de PureBasic ! Ca compte !
- **Gaetan Dupont-Panon** : Pour le fabuleux nouvel éditeur visuel de fenêtres, qui fonctionne sour Windows, Linux et OS X !
- **Jean R. VIALE** : Pour avoir amélioré et corrigé de manière conséquente l'aide française, et pour continuer à le faire !

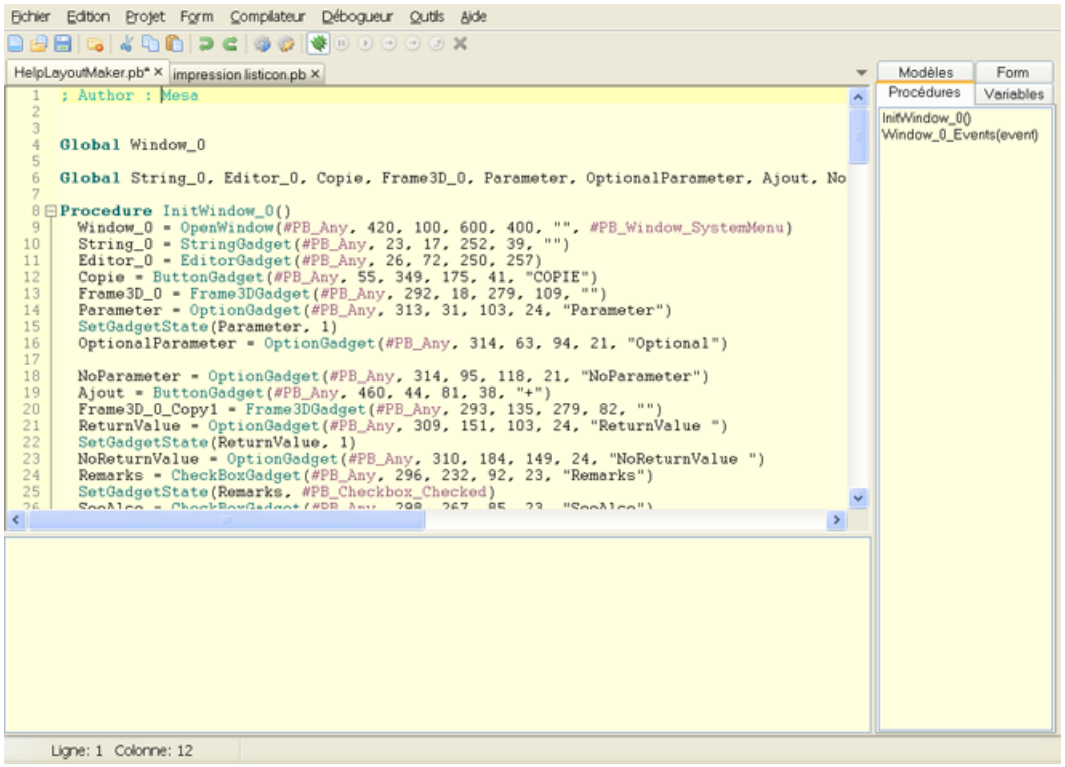
Deuxième partie

L'IDE de PureBasic

Chapitre 8

Débuter avec l'IDE

L'IDE PureBasic est un environnement complet qui permet la création et l'édition de fichiers sources, leur compilation, débogage ainsi que la création de l'exécutable final. Il sert d'interface au compilateur ainsi qu'au débogueur .
La fenêtre principale de l'IDE se divise en 3 parties :



La zone d'édition du code (en dessous de la barre d'outils)
Tous les codes sources sont affichés ici. Il est possible de passer de l'un vers l'autre à l'aide des onglets situés entre la barre d'outils et la zone d'édition.
La palette d'outils (par défaut située à droite de la zone d'édition)
La palette contient plusieurs outils rendant la programmation plus facile et plus productive. Les outils affichés ici peuvent être configurés, déplacés ou même enlevés si besoin est. Voir Configurer l'IDE pour plus de renseignements.
Le rapport d'activité (situé en dessous de la zone d'édition)
Les erreurs de compilation ainsi que les messages d'erreur du débogueur seront affichés dans cette partie. Il peut être visible ou caché séparément pour chaque fichier.

Le reste de l'interface se compose du menu principal et de la barre d'outils. Cette dernière est

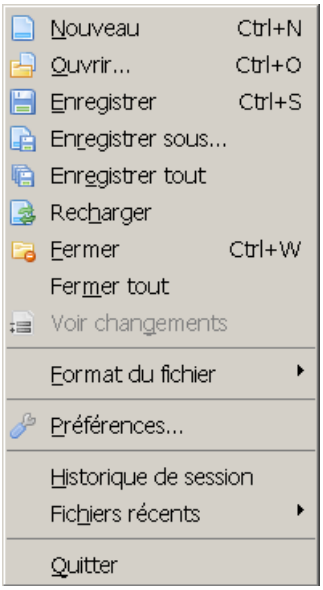
simplement un raccourci des fonctions disponibles dans le menu et peut être complètement paramétrée (chaque fonction du menu peut être enlevée, déplacée ou ajoutée à la barre d'outils). Pour savoir ce que fait un bouton, il suffit de laisser le pointeur de la souris dessus un court instant et une aide apparaîtra (et affichera la fonction du menu correspondante). Les commandes du menu sont expliquées dans une autre section.

Chapitre 9

Gestion des fichiers

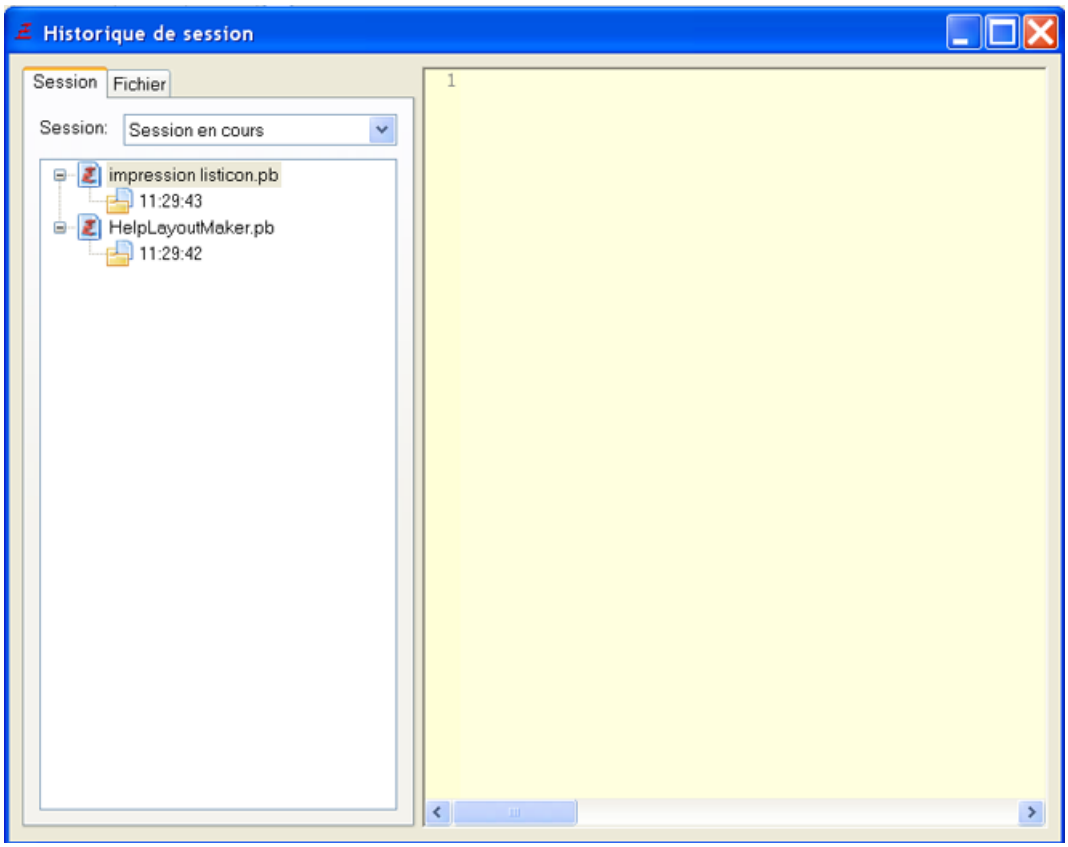
Le menu ”Fichier” permet de réaliser les opérations de base tel que l'ouverture ou l'enregistrement des fichiers.
Il est possible d'éditer plusieurs codes sources simultanément, et de passer de l'un à l'autre grâce aux onglets situés sous la barre d'outils. Par défaut, les raccourcis claviers Ctrl+Tab et Ctrl+Shift+Tab permettent de passer au fichier suivant (ou précédent).
L'IDE permet l'édition de fichiers texte qui ne sont pas du code source. Dans ce mode ”texte brut”, les caractéristiques liées au code tels que la coloration synthaxique, la correction des cas, l'auto-completion automatique sont désactivées. Lors de l'enregistrement de ces fichiers, l'IDE n'ajoutera pas ses paramètres à la fin du fichier, même si cela est configuré pour les fichiers de code dans les préférences .
Un fichier qui contient du code ou non dépend de son extension. Les extensions de fichiers standard PureBasic (pb, pbi et pbf) sont reconnues comme des fichiers contenant du code. Plusieurs extensions de fichier peuvent être reconnues comme des fichiers de code en les configurant dans la section ”Editeur” des Préférences .

Contenu du menu ”Fichier”



Nouveau
Créé un nouveau code source vide.
Ouvrir
Ouvre un code source existant pour l'éditer.
Les fichiers de type 'texte' seront tous chargés dans la zone d'édition. Il est aussi possible d'ouvrir des fichiers binaires et dans ce cas ils seront ouverts par le visualisateur de fichier interne.

Enregistrer
Enregistre le source en cours d'édition sur le disque. S'il n'a pas encore été enregistré, son nom et son emplacement devront être définis à l'aide de la fenêtre de dialogue, sinon le code sera enregistré dans le fichier précédent.
Enregistrer sous
Enregistre le code source en cours d'édition à un emplacement différent. Une boîte de dialogue demandant le nom du nouveau fichier sera affichée. L'ancien code source restera inchangé.
Enregistrer tout
Enregistre tous les codes source ouverts.
Recharger
Recharge le fichier courant à partir du disque. Cela annulera tous les changements non sauvegardés.
Fermer
Ferme le code source en cours d'édition. Si c'était le dernier fichier, l'IDE affichera un nouveau fichier vide.
Fermer tout
Ferme tous les codes source en cours d'édition. L'IDE affichera un nouveau fichier vide.
Voir changements
Affiche les modifications qui ont été apportées au code source courant par rapport à la version qui existe sur le disque dur.
Format du fichier
Dans ce sous-menu, il est possible de sélectionner le format d'encodage ainsi que les terminaisons de lignes du fichier courant. L'IDE supporte les fichiers textes ASCII ou UTF-8. Les terminaisons de ligne supportées sont : Windows (CRLF), Linux/Unix (LF) et MacOSX (CR). Les paramètres par défaut appliqués lors de la création d'un nouveau fichier peuvent être changés dans les préférences de l'IDE.
Préférences
Cette fenêtre permet de contrôler le look et le comportement de l'IDE pour qu'il s'adapte aux besoins de chacun. Pour une description détaillée des fonctions disponibles, voir Configurer l'IDE .
Historique de session
L'historique de session est un outil puissant qui enregistre régulièrement les modifications apportées aux fichiers ouverts dans l'IDE, dans une base de données. Une session est créée lors du lancement de l'IDE, et est fermée lorsque l'IDE se ferme. Cette option est utile pour revenir à une version antérieure d'un fichier ou pour retrouver un fichier supprimé ou corrompu. C'est un peu comme un outil de sauvegarde des sources mais limité dans le temps (par défaut, un mois d'enregistrement). Il n'est pas destiné à remplacer un vrai système de gestion des sources comme SVN ou GIT. Le code source sera stocké sans chiffrement, donc si vous travaillez sur un code source sensible, assurez-vous d'avoir ce fichier de base de données dans un endroit sûr, ou désactiver cette fonction.
Pour configurer l'outil historique de la session, voir préférences .



Fichiers récents
Le sous-menu affiche la liste des dix derniers fichiers ouverts. Sélectionner un fichier dans le sous-menu l'ouvrira à nouveau.

Quitter
Cela permet bien entendu de quitter l'IDE. Si un ou plusieurs fichiers ne sont pas enregistrés, une confirmation pour chacun des fichiers sera demandée.

Chapitre 10

Edition du code

L'IDE du PureBasic se comporte comme un éditeur de texte classique en ce qui concerne la saisie de texte. Les touches de curseur, Page précédente, Page suivante, Début et Fin permettent de se déplacer dans le code. Ctrl+Début se rend au début du code et Ctrl+Fin se déplace à la fin.
Les raccourcis par défaut Ctrl+C (copier), Ctrl+X (couper) et Ctrl+V (coller) servent à l'édition. La touche Insérer change le mode d'édition pour déterminer si le texte doit etre remplacé ou ajouté (le mode par défaut est l'ajout). La touche Supprimer supprime une lettre à droite du curseur. Maintenir la touche Shift appuyée et utiliser les flèches permet de sélectionner du texte.
Au delà des fonctions de base, l'IDE a de nombreuses fonctionnalités dédiées à la programmation PureBasic.

Indentation

Quand un retour à ligne est effectué, l'indentation (nombre d'espaces/tabulations en début de ligne) de la ligne courante et de la ligne suivante sera calculée automatiquement en fonction du mot-clé qui est présent sur la ligne. Un mode "block" est aussi disponible. Dans ce cas, la nouvelle ligne aura la même indentation que la ligne courante. Tout ceci est paramètrable dans les preférences .

Caractères de tabulation

Par défaut, l'IDE n'insère pas une vraie tabulation quand la touche Tab est utilisée. En cas d'échange de source, ceci permet d'éviter les problèmes d'affichage inhérents aux normes de tabulation utilisées.
A la place, deux caractères espaces sont insérés. Ce comportement peut être modifié dans les préférences de l'IDE (voir Configurer l'IDE).

Comportement particulier de la touche Tab

Quand la touche Tab est utilisée alors que la sélection est vide ou ne contient que quelques caractères, elle agit comme décrit ci-dessus (insertion d'un nombre d'espaces, ou d'une vraie tabulation selon le réglage défini dans les préférences de l'IDE).
Quand la touche Tab est utilisée alors qu'une ou plusieurs lignes complètes sont sélectionnées, des espaces (ou des tabulations, en fonction de la configuration) sont insérés au début de chaque ligne. Cela permet d'indenter rapidement des blocs complets de code.
Quand la combinaison de touches Shift+Tab est utilisée alors que plusieurs lignes sont sélectionnées, les espaces/tabulations au début de chaque ligne sont supprimés afin de réduire l'indentation du bloc complet.

Retrait (Indentation) / Alignement des commentaires :

Similaire au comportement de l'onglet spécial ci-dessus, les raccourcis clavier Ctrl+E et Ctrl+Maj+E (CMD+E et CMD+Maj+E sur OSX) peuvent être utilisés pour modifier le retrait des commentaires sélectionnés. Cela aide à aligner les commentaires à la fin du code, pour le rendre encore plus lisible. Les raccourcis utilisés peuvent être configurés dans les préférences .

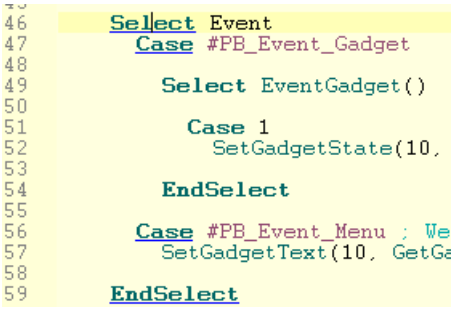
Sélection des blocs de code :

Le raccourci Ctrl+M (CMD+M sur OSX) peut être utilisé pour sélectionner le bloc de code qui contient la position du curseur (par exemple un bloc 'If', une boucle ou une procédure). Une utilisation répétée du raccourci sélectionne d'autres blocs de code environnant.
Le raccourci Ctrl+Maj+M (CMD+Maj+M sur OSX) a la fonction inverse et rétablit la sélection du bloc précédemment sélectionné avec Ctrl+M.
Les raccourcis utilisés peuvent être configurés dans les préférences .

Double-cliquer sur un mot

En principe, double-cliquer sur un mot le sélectionne. Néanmoins, dans quelques cas, le comportement est différent :
Maintenir la touche Ctrl appuyée et double-cliquer sur le nom d'une procédure , définie dans le même fichier, permet de se rendre directement à sa déclaration. Si la procédure se trouve dans un autre fichier, celui-ci doit déjà être ouvert dans l'IDE.
Double-cliquer sur un mot-clé IncludeFile ou XincludeFile ouvrira le fichier dans l'IDE (le nom du fichier doit être spécifié en entier, sans utiliser de constante par exemple).
De la même manière, double-cliquer sur le mot-clé IncludeBinary aura pour effet d'ouvrir le fichier dans le visualisateur de fichier intégré à l'IDE.

Marqueur de parenthèse et de mots-clés :

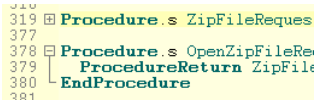


Quand le curseur est sur une parenthèse ouvrante ou fermante, l'IDE mettra en surbrillance l'autre parenthèse qui lui correspond. Si aucune parenthèse ne correspond (ce qui est une erreur de syntaxe en PureBasic), l'IDE affichera la parenthèse en rouge. Le même concept est appliqué aux mots-clés. Si le curseur est sur un mot-clé comme "If", l'IDE soulignera ce mot-clé et tous les autres mots-clés qui lui correspond, comme "Else", "ElseIf" ou "EndIf". Si il manque des mots-clés, il sera souligné en rouge. Le menu "Aller au mot-clé correspondant" décrit ci-dessous peut être utilisé pour aller rapidement d'un mot-clé à l'autre.
Le marqueur de parenthèse et de mots-clés est configurable dans les préférences .

Aide syntaxique dans la barre d'état

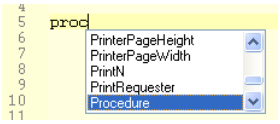
L'IDE affichera les paramètres nécessaires pour chaque fonction PureBasic qui est en cours de frappe. Cela rend plus facile la saisie, en montrant les paramètres qui sont indispensables. Cela fonctionne aussi pour les procédures , prototypes , interfaces ou fonctions importées , s'ils sont déclarés dans le même fichier source ou dans le même projet .

Options de pliage



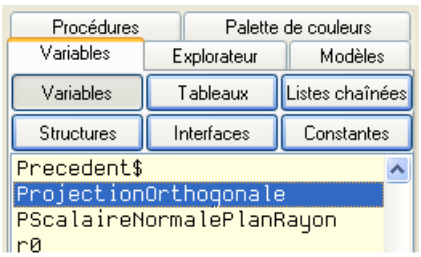
Quand un mot-clé de pliage est rencontré (Procedure / EndProcedure par défaut, d'autres peuvent être ajoutés) l'IDE marque la région délimitée par ces deux mots-clés par un [-] sur la ligne du premier mot et une ligne verticale jusqu'à la fin du bloc.
En cliquant sur le [-], il est possible de cacher (replier) le contenu du bloc pour garder une meilleure vue d'ensemble sur les codes sources conséquents. Le [-] se transformera alors en [+]. En cliquant de nouveau dessus, le code sera de nouveau affiché (déplié).
Note : Même si l'état des blocs est préservé lors de l'enregistrement du fichier, le fichier lui-même contient bien sûr toutes les lignes de codes. Cette option affecte seulement l'affichage du code source, pas son contenu.
Deux autres délimiteurs de replis sont activés par défaut : ";{" et ";}". Comme ";" marque le début d'un commentaire en PureBasic, ils seront ignorés par le compilateur. Néanmoins, ils permettent de placer des délimiteurs à des endroits arbitraires qui ne correspondent pas forcément à un mot-clé PureBasic.

Auto-complétion



Pour ne pas avoir à se souvenir de tous les noms des commandes PureBasic, il y a l'auto-complétion qui peut rendre bien des services.
Après avoir tapé le début du nom de la commande, une liste des commandes possibles qui débute par ce mot s'affichera. Une liste de choix est aussi affichée après avoir tapé "\" sur une variable liée à une structure ou une interface.
Une fois le mot sélectionné à l'aide des flèches il peut être inséré dans le code source en appuyant sur la touche Tab. Il est aussi possible de continuer à taper les lettres pendant que la liste est affichée, pour réduire le choix. Si plus aucune proposition n'est disponible, alors la boîte se fermera automatiquement. La touche Echappement permet de fermer la liste à tout moment. La liste se ferme aussi automatiquement si un bouton souris est utilisé dans l'IDE.
Note : Pour configurer le contenu qui sera affiché dans la liste d'auto-complétion, ainsi que ses nombreuses fonctionnalités, voir Configurer l'IDE . Par exemple, il est possible de désactiver l'apparition automatique de la liste (Ctrl+Espace sera alors necessaire pour l'afficher).

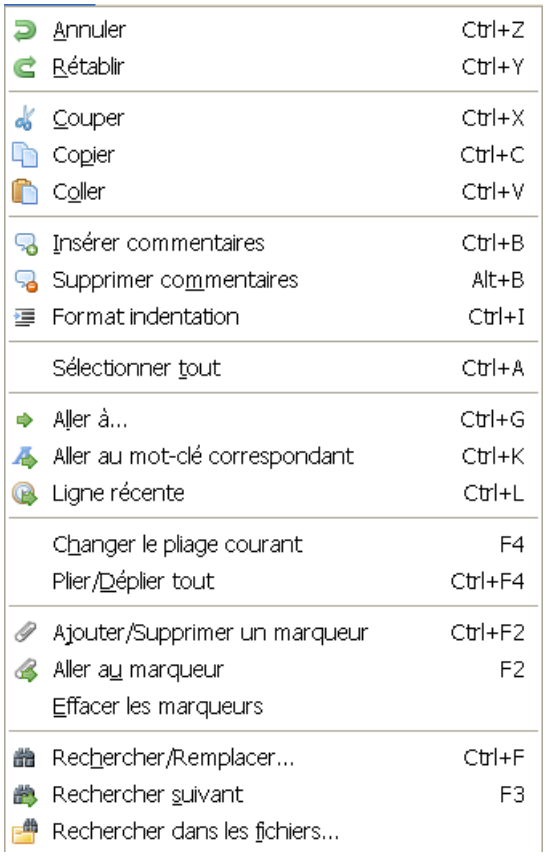
Panneau d'outils



La plupart des outils qui rendent la navigation et l'édition de code sources plus aisés peuvent être

ajoutés au panneau d'outils situé sur le côté de la zone d'édition. Pour avoir un aperçu des outils disponibles et pour les configurer, voir Les outils intégrés .

Le menu édition



Les rubriques suivantes expliquent les commandes disponibles dans le menu édition. A noter que bon nombre de ces commandes sont aussi présentes dans le menu contextuel de la zone d'édition.

Annuler
Annule la dernière action effectuée dans la zone d'édition. Il est possible d'annuler plusieurs actions à la suite.

Rétablir
Refait la dernière action annulée par la commande "Annuler".

Couper
Copie le texte sélectionné dans le presse-papier et le supprime du code.

Copier
Copie le texte sélectionné dans le presse-papier.

Coller
Insère le contenu du presse-papier dans le code, à la position du curseur. Si une partie du code était sélectionnée, elle sera alors remplacée par le contenu du presse-papier.

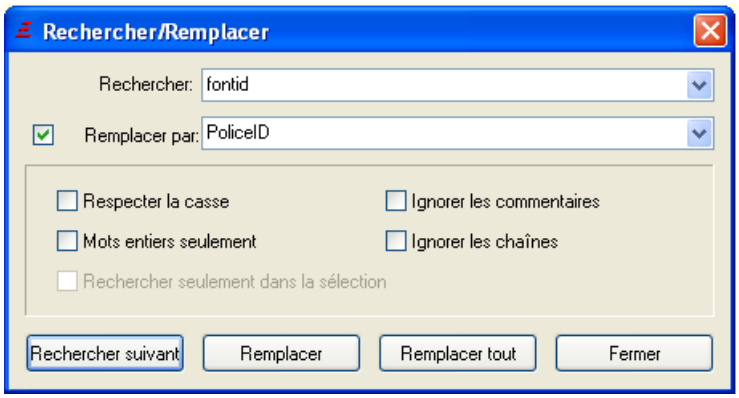
Insérer commentaires
Insère un commentaire (";") au début de chaque ligne sélectionnée. Cela permet de commenter rapidement un nombre conséquent de lignes.

Supprimer commentaires
Enlève le commentaire (";") du début de chaque ligne sélectionnée. Cela permet de dé-commenter rapidement un nombre conséquent de lignes. C'est la fonction inverse de "Insérer commentaire", mais elle fonctionne aussi sur des commentaires entrés manuellement.

Format et indentation
Reformate le code source sélectionné pour aligner le code en fonction des règles définies dans les préférences .

Sélectionner tout
Sélectionne le code source en entier.

Aller à
Permet de se rendre directement à une ligne donnée.
Aller au mot-clé correspondant
Si le curseur est actuellement sur un mot-clé comme "If", le curseur ira directement sur le mot-clé correspondant (dans ce cas "EndIf").
Ligne récente
L'IDE garde une trace des lignes qui sont visualisées. Par exemple, si la position du curseur est changée par la commande "Aller à" ou en cliquant sur l'outil Navigateur de procédures , cette fonction permet de revenir rapidement à la ligne précédente. L'IDE retient 20 positions différentes.
A noter que l'enregistrement ne se fait que si le déplacement est conséquent (ex : pas si les touches haut/bas sont utilisées).
Changer le pliage courant
Change l'état du point de repli associé à la ligne courante (le déplier ou le replier le cas échéant).
Plier/Déplier tout
Change l'état de tous les points de replis dans le code source, très utile pour cacher toutes les procédures d'un code source (ou pour ré-afficher le code complet en une seule fois).
Ajouter/Supprimer un marqueur
Cette commande permet d'ajouter un marqueur sur la ligne courante (ou de le retirer si il y en avait déjà un). Un marqueur permet de retrouver rapidement une ligne de code. Il est identifié par une petite flèche horizontale (verte, par défaut) à côté de la ligne concernée. Il est possible de passer de marqueur en marqueur en utilisant la commande "Marqueur suivant".
Note : Pour ajouter/retirer un marqueur à l'aide de la souris, il faut maintenir la touche Ctrl appuyée pendant le clic sur la bordure qui affiche les marqueurs (la colonne de numérotation des lignes n'est pas prise en compte).
Aller au marqueur
Se rend au premier marqueur disponible, situé en dessous de la ligne en cours d'édition. Si il n'y a plus de marqueur en dessous de cette ligne, la recherche reprend au début du code source. En utilisant le raccourci "Marqueur suivant" (F2, par défaut) plusieurs fois, il est possible de parcourir tous les marqueurs du code très rapidement.
Effacer les marqueurs
Retire tous les marqueurs du code source en cours d'édition.
Rechercher/Remplacer



Cette fenêtre permet de rechercher un mot (ou une suite de mots) dans le code source et de les remplacer par d'autres mots si nécessaire.
Le bouton "Chercher suivant" démarre la recherche. La recherche peut continuer une fois un terme trouvé en utilisant la commande du menu "Rechercher à nouveau" (raccourci F3 par défaut).
Pour rendre la recherche plus pointue, les options suivantes sont disponibles :
Respecter la casse : Seul le texte qui est strictement identique au mot recherché sera retenu (les minuscules ne sont pas identiques aux majuscules).
Mots entiers seulement : Recherche seulement les termes qui forment un mot. Les termes qui contiennent le mot recherché ne seront pas retenus.
Ignorer les commentaires : Tous les commentaires PureBasic sont exclus de la recherche.
Ignorer les chaînes : Tous les contenus des strings (entre " ") sont ignorés.
Rechercher seulement dans la sélection : recherche seulement dans le texte sélectionné. C'est une option très utile quand elle est utilisée en conjonction avec la fonction "Remplacer tout". Dans ce cas, le remplacement ne s'opèrera uniquement que dans la sélection.

En activant la case à cocher ”Remplacer par”, la fonction remplacer devient disponible. ”Chercher suivant” cherchera toujours l'occurrence suivante, et chaque clic sur le bouton ”Remplacer” remplacera le texte trouvé par le contenu du champ ”Remplacer par”.
En cliquant sur ”Remplacer tout”, tous les termes correspondants à la recherche situés en dessous du curseur seront remplacés (sauf si l'option ”Rechercher seulement dans la sélection” est activée).

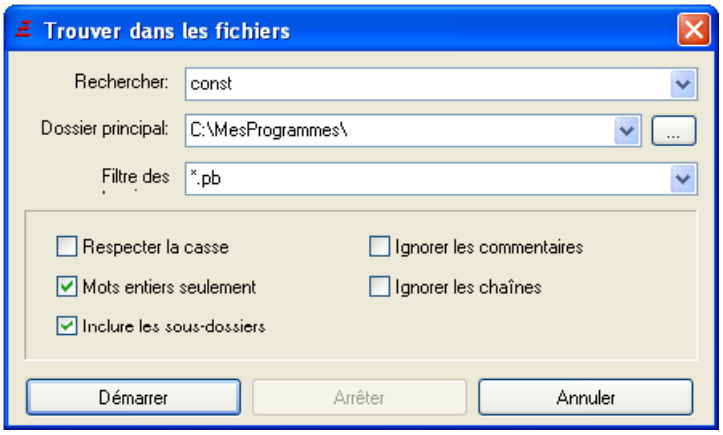
Rechercher suivant

Continue la recherche correspondant à la dernière recherche entamée dans la fenêtre ”Chercher/Remplacer”.

Rechercher précédent

Continue la recherche en remontant à la recherche précédente entamée dans la fenêtre ”Chercher/Remplacer”.

Rechercher dans les fichiers



Cette fonction permet de rechercher parmi tous les fichiers d'un répertoire donné (et optionnellement dans ses sous-répertoires).
Il suffit de spécifier le mot (ou la séquence de mots) à rechercher ainsi que le répertoire de base (champ ”Répertoire”). Il est possible d'indiquer des filtres pour inclure uniquement les fichiers voulus (ex : *.pb). Plusieurs filtres peuvent être spécifiés, en les séparant par une virgule ”,” (un filtre vide ou *.* recherche dans tous les fichiers). Des options similaires à la fenêtre ”Rechercher/Remplacer” sont disponibles pour rendre la recherche plus pointue.
La case à cocher ”Inclure les sous-répertoires” permet de rechercher dans tous les sous-répertoires (d'une manière récursive) du répertoire de base.
Quand la recherche débute, une fenêtre indépendante sera ouverte pour afficher les résultats de la recherche en indiquant le fichier, le numéro de la ligne ainsi que le contenu de la ligne.
En double-cliquant sur une des lignes, le fichier sera automatiquement ouvert dans l'IDE à la ligne concernée.

Chapitre 11

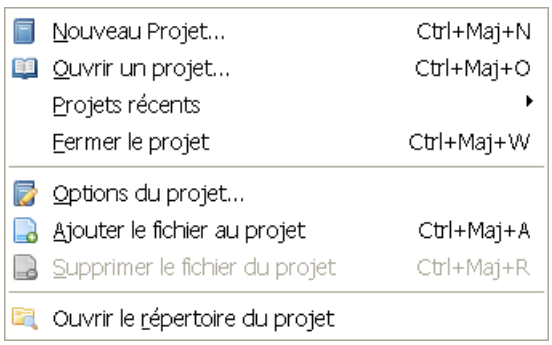
Gestion de projets

L'IDE contient des fonctionnalités avancées pour gérer de larges projets. Ces fonctionnalités sont complètement facultatives, les programmes peuvent être créés et compilés sans recourir à la création d'un projet. Cependant, dès qu'un programme atteint une certaine taille et qu'il se décompose en de nombreux fichiers, il sera probablement plus facile de le gérer au sein d'un projet.

Aperçu de la gestion de projets

Un projet regroupe tous les codes sources et les autres fichiers nécessaires au programme à un seul endroit, avec un accès rapide à tous les fichiers à l'aide de l'outil de projet . Les fichiers sources inclus dans le projet peuvent être scannés automatiquement pour l'auto-complétion même s'ils ne sont pas ouverts dans l'IDE. Donc les fonctions, constantes, variables etc. du projet entier peuvent être utilisés dans l'auto-complétion. Le projet enregistre aussi les fichiers ouverts lors sa fermeture, donc l'espace de travail sera retabli à l'identique lors de la réouverture du projet.
De plus, le projet regroupe toutes les options de compilation à un seul endroit (dans le fichier du projet) et permet même de créer plusieurs cibles de compilation par projet. Une cible de compilation est un ensemble d'options permettant de générer plusieurs programmes distincts, ou plusieurs versions différentes du même programme. Toutes les cibles peuvent être compilées en une seule étape, ce qui permet de gagner en productivité.
Pour compiler un projet à partir d'un script ou d'un makefile, l'IDE accepte une option en ligne de commande pour compiler un projet sans ouvrir d'interface. Voir la section Commutateurs de la ligne de commande pour plus de détails.
Tous les noms de fichiers et les chemins d'un projet sont stockés de manière relative à son fichier de configuration, donc il est facile de déplacer ou de partager un projet, tout en gardant la structure du répertoire intacte.

Le menu Projet



Nouveau projet

Crée un nouveau projet. Si un autre projet était déjà ouvert, il sera fermé. La fenêtre d'options sera ouverte où le nom du projet devra être saisi. La configuration du projet se fera à partir de cette fenêtre.

Ouvrir un projet

Ouvre un projet existant. Si un autre projet était déjà ouvert, il sera fermé. Les fichiers qui étaient déjà ouverts dans ce projet seront automatiquement ré-ouverts (si cette option est activée).

Projets récents

Ce sous-menu affiche la liste des projets les plus récemment ouverts.

Fermer le projet

Ferme le projet actuellement ouvert. La configuration du projet sera enregistrée et les fichiers ouverts associés au projets seront fermés (si cette option est activée).

Options du projet

Ouvre la fenêtre d'options propre au projet (voir ci-dessous pour plus d'informations)

Ajouter le fichier au projet

Ajoute le fichier courant dans le projet courant. Les fichiers qui appartiennent à un projet ont le symbole ">" devant leur nom dans les onglets.

Retirer le fichier du projet

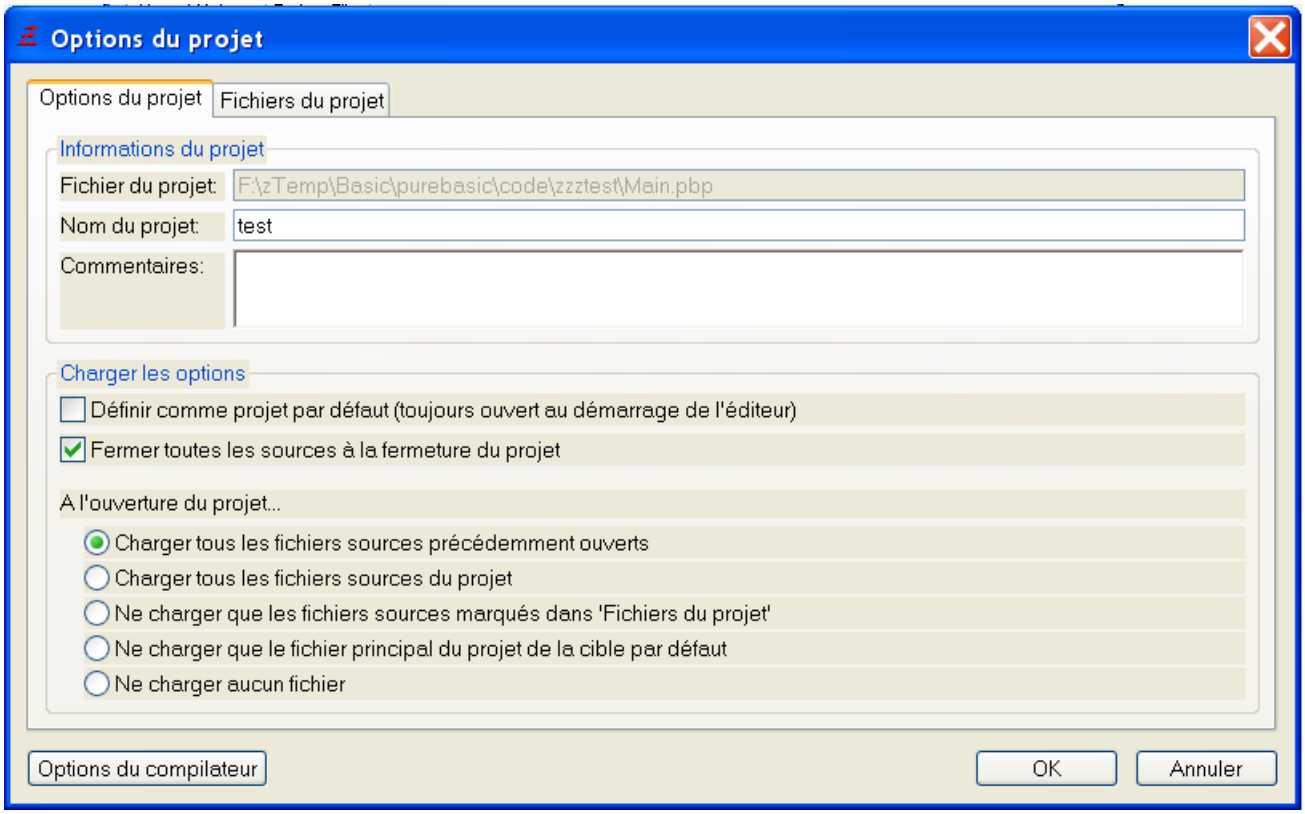
Retire le fichier courant du projet courant.

Ouvrir le répertoire du projet

Ouvre le répertoire qui contient le fichier projet dans l'explorateur de fichier du système.

La fenêtre d'options du projet

La fenêtre d'options du projet est le centre de configuration du projet. Les paramètres relatifs au projet et aux fichiers gérés par le projet sont accessibles ici.



Les paramètres suivants peuvent être modifiés dans l'onglet "Options du projet" :

Nom de fichier du projet

Affiche le nom de fichier du projet. Il ne peut être défini seulement pendant la création du projet.

Nom du projet

Nom du projet. Ce nom sera affiché dans la barre de titre de l'IDE et dans le menu "Projets récents".

Commentaires

Ce champ permet d'ajouter des informations complémentaires sur le projet. Elles seront affichées dans

l'onglet d'information du projet.

Définir comme projet par défaut

Le projet par défaut sera ouvert automatiquement à chaque démarrage de l'IDE. Un seul projet peut être le projet par défaut. S'il n'y a pas de projet par défaut, l'IDE chargera le dernier projet qui était ouvert lors de sa fermeture (s'il y en avait un).

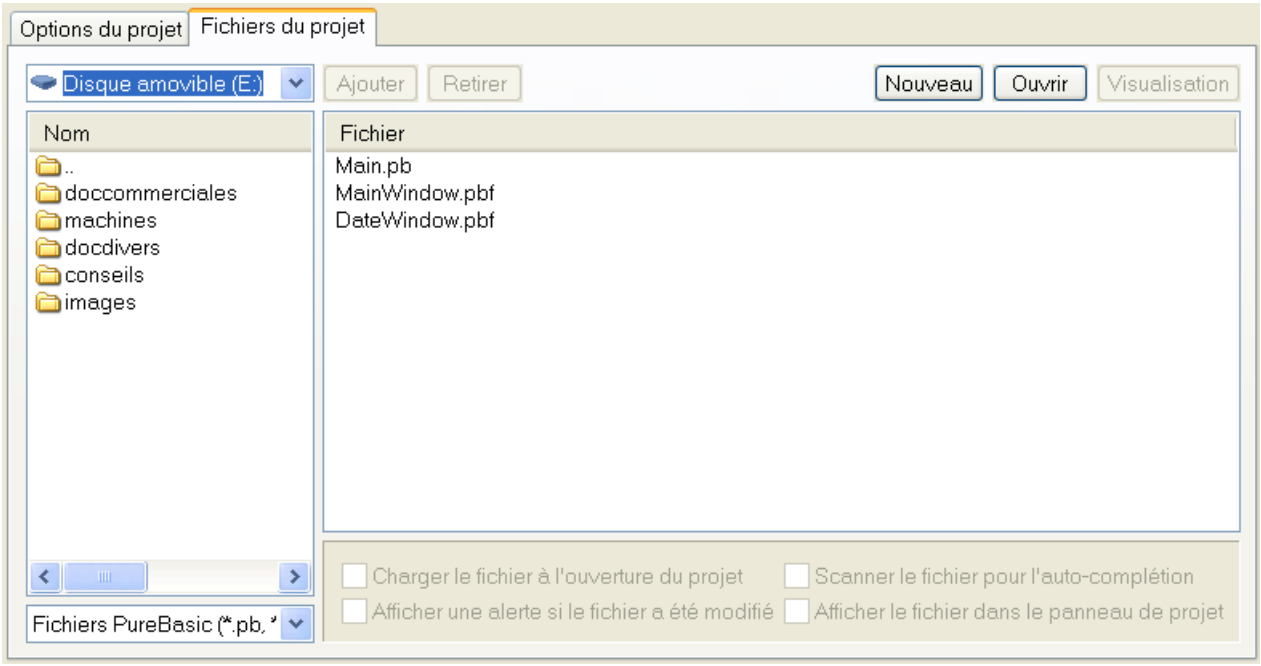
Fermer tous les fichiers lors de la fermeture du projet

Permet de fermer tous les fichiers qui appartiennent au projet lorsque ce dernier est fermé.

A l'ouverture du projet...

> **Charger tous les fichiers sources précédemment ouverts**
> Quand le projet est ouvert, tous les fichiers sources précédemment ouverts seront chargés.
> **Charger tous les fichiers sources du projet**
> Quand le projet est ouvert, tous les fichiers sources du projet seront chargés.
> **Ne charger que les fichiers sources marqués dans 'Fichiers du projet'**
> Quand le projet est ouvert, seuls les fichiers sources marqués dans 'Fichier du projet' seront chargés. De cette manière, il sera possible d'avoir toujours la même configuration lorsque qu'un projet est ouvert.
> **Ne charger que le fichier principal du projet de la cible par défaut**
> Quand le projet est ouvert, seul le fichier principal de la cible par défaut sera chargé.
> **Ne charger aucun fichier**
> Aucun fichier source ne sera chargé lors de l'ouverture du projet.

L'onglet "Fichiers du projet" affiche la liste des fichiers qui compose le projet et permet de changer leurs paramètres. L'explorateur sur la partie gauche permet de rajouter facilement des fichiers au projet.



Les boutons en haut de l'onglet ont les fonctions suivantes :

> **Ajouter**
> Ajouter le(s) fichier(s) sélectionné(s) de l'explorateur dans le projet.
> **Retirer**
> Retire du projet les fichiers sélectionnés dans la liste.
> **Nouveau**
> Ouvre une fenêtre permettant de choisir le nom du nouveau fichier à créer. Ce fichier sera créé, ajouté au projet et ouvert dans l'IDE.
> **Ouvrir**
> Ouvre une fenêtre permettant de choisir un fichier existant à ajouter au projet. Le fichier sera automatiquement ouvert dans l'IDE.
> **Voir**
> Ouvre le(s) fichier(s) sélectionné(s) de la liste dans l'IDE, ou dans le visualisateur de fichiers si ce sont des fichiers binaires.

Les cases à cocher en dessous de la liste des fichiers représentent les options disponibles pour chaque fichier. Elles peuvent être changées pour un seul ou plusieurs fichiers à la fois. Voici leurs descriptions :

Changer le fichier à l'ouverture du projet
Indique à l'IDE de charger ce fichier lorsque que le projet est ouvert et que l'option "Ne charger que les fichiers sources marqués dans 'Fichiers projet"' est activée dans l'onglet "Options du projet".

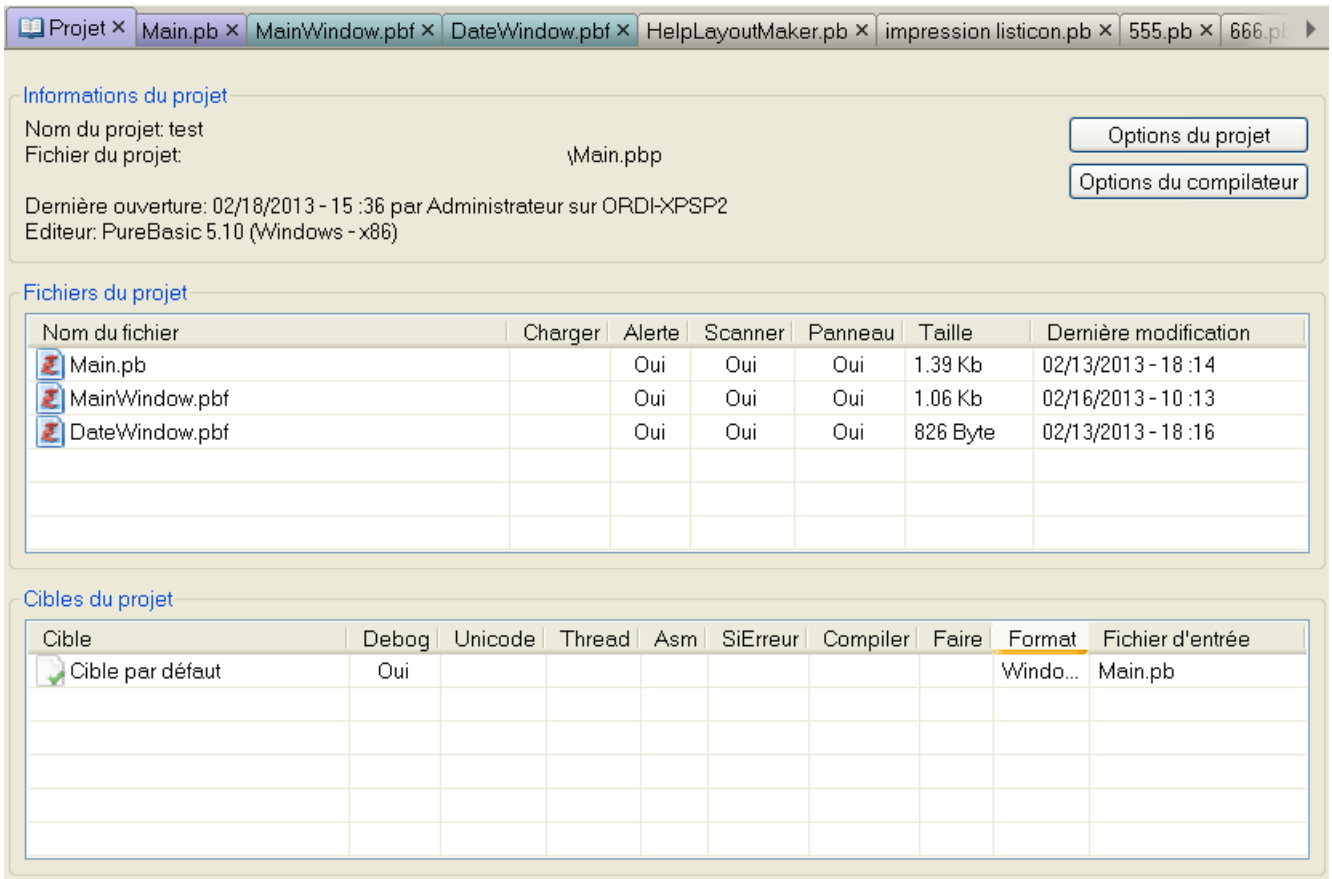
Afficher une alerte si le fichier a été modifié
Quand le projet est fermé, l'IDE calcule un checksum de tous les fichiers qui ont cette option activée, et affichera une alerte si le fichier a été modifié en dehors de l'IDE. Cela permet de savoir qu'une modification externe a eu lieu lorsque l'on partage des fichiers avec différents projets. Cette option ne devrait pas être activée pour les fichiers binaires imposants pour garder une bonne rapidité d'ouverture et de sauvegarde du projet.

Scanner le fichier pour l'auto-complétion
Active le scan du fichier pour la liste d'auto-complétion, même si ce dernier n'est pas chargé dans l'IDE. Cette option est activée par défaut pour tous les fichiers non-binaires. Il faudra le désactiver pour tous les fichiers qui ne contiennent pas de code source, ainsi que pour les fichiers qui n'ont pas leurs places dans la liste d'auto-complétion.

Afficher le fichier dans le panneau de projet
Affichera le fichier dans le panneau de projet, qui se trouve à droite de l'éditeur. Si le projet a beaucoup de fichiers, il peut être utile d'en cacher quelques-un pour garder un accès rapide aux fichiers principaux.

Résumé du projet

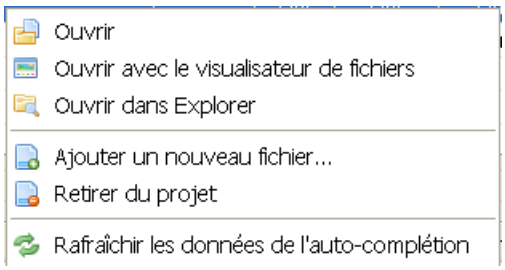
Quand un projet est ouvert, le premier onglet des fichiers affiche un résumé du projet et de ses fichiers.



Informations du projet
Cette section affiche les informations générales du projet, comme le nom du projet, ses commentaires ou quand le fichier a été ouvert pour la dernière fois.

Fichiers du projet

Cette section affiche tous les fichiers du projet et leurs paramètres. Un double-clic sur un de ces fichiers l'ouvre automatiquement dans l'IDE. Un clic-droit fait apparaître un menu contextuel avec davantage d'options :



Ouvrir - Ouvre le fichier dans l'IDE.
Ouvrir avec le visualisateur de fichiers - Ouvre le fichier dans le visualisateur intégré de l'IDE.
Ouvrir avec l'explorateur - Ouvre le fichier dans l'explorateur du système d'exploitation.
Ajouter un nouveau fichier - Ajoute un nouveau fichier au projet.
Retirer du projet - Retire le(s) fichier(s) sélectionné(s) du projet.
Rafraîchir les données de l'auto-complétion - Rescanne tous les fichiers du projet pour actualiser les données de l'auto-complétion.

Cibles du projet
Cette section affiche toutes les cibles du projet et quelques un de leurs paramètres. Un double-clic sur l'une des cibles l'ouvre dans les options de compilation . Un clic-droit affiche un menu contextuel avec davantage d'options :

Editer la cible - Ouvre la cible dans la fenêtre options de compilation.
Définir comme défaut - Défini cette cible comme celle par défaut.
Inclure dans 'Créer toutes les cibles' - Inclut cette cible dans 'Créer toutes les cibles'.

La panneau projet

Un outil intégré permet d'accéder rapidement aux fichiers du projet. Pour plus d'informations voir la section outils intégrés .

Chapitre 12

Compilation d'un programme

La compilation d'un programme est facile, il suffit de sélectionner 'Compiler/Executer' (raccourcis F5 par défaut) et le code sera compilé et exécuté.
Les "Options de compilation" permettent de paramétrer le processus de compilation. Les choix effectués sont associés au fichier ou au projet courant et sont persistants, même quand ils sont fermés. Par défaut ces paramètres sont écrits à la fin du fichier, sous forme de commentaires (invisibles quand le fichier est chargé par l'IDE). Il est possible de changer cet emplacement si, par exemple, un outil de gestion de versions tel que CVS est utilisé (pour éviter les conflits entre les utilisateurs qui auront chacun leurs propres paramètres en fin de fichier).
Si une erreur est rencontrée lors de la compilation, un message sera affiché en indiquant la ligne incriminée. Ce message sera aussi ajouté dans le rapport d'activité.

Le menu "Compilateur"

Compiler/Exécuter
Lance la compilation du code source en cours d'édition en tenant compte de ses options de compilation. Le programme est créé dans un répertoire temporaire, mais il sera exécuté comme si il avait été lancé à partir du répertoire du code source (donc charger un fichier de ce répertoire fonctionnera).
Le code source n'a pas besoin d'être enregistré pour être compilé (mais les fichiers qu'il inclut doivent être enregistrés).
La commande "Compiler/Exécuter" utilise le paramétrage du débogueur (actif ou inactif), qu'il soit défini à partir du menu ou des options de compilations (les deux reviennent au même).
Exécuter
Exécute le dernier programme compilé une nouvelle fois. Tout changement concernant le débogueur (actif ou inactif) ne sera pas pris en compte par cette commande.
Vérification de la syntaxe

Vérifie la syntaxe du code.

Compiler avec le Débogueur

Lance la compilation de la même manière que la commande "Compiler/Exécuter", en forçant l'utilisation du débogueur. Cela permet de tester rapidement une compilation avec le débogueur lorsqu'il est normalement désactivé.

Compiler sans le Débogueur

Lance la compilation de la même manière que la commande "Compiler avec le débogueur", en désactivant le débogueur.

Redémarrer le compilateur (Windows uniquement)

Relance le compilateur et réinitialise toutes les bibliothèques et résidents chargées (sous Windows, le compilateur se lance lors du démarrage de l'IDE et se met en attente pour une plus grande rapidité). Cela a pour effet de mettre à jour la liste des fonctions, structures, interfaces et constantes reconnues par l'IDE. Cette commande permet de charger une nouvelle bibliothèque utilisateur nouvellement installée sans redémarrer l'IDE. De plus, pour les développeurs de bibliothèques, cela facilite le test en évitant de relancer l'IDE.

Options du Compilateur...

Ouvre la fenêtre d'options, qui permet d'ajuster les paramètres concernant la compilation du code source en cours d'édition.

Créer un éxecutable...

Ouvre une boîte de dialogue de sauvegarde, demandant le nom de l'exécutable à créer. Si le format exécutable est réglé sur DLL, il créera une DLL sous Windows, un objet partagé sous Linux et une dylib sous OS X. Lorsque vous créez un fichier exécutable sous OS X, en ajoutant '. App 'au nom de l'exécutable créera un fichier exécutable fourni avec la structure de répertoires nécessaires, y compris l'icône. Si aucun '. App' n'est défini, il va créer un exécutable standard de type console.

Cible par défaut

Quand un projet est ouvert, ce sous-menu permet de changer rapidement de cible par défaut. La cible par défaut est celle qui sera lancée automatiquement par le menu "Compiler/Exécuter.

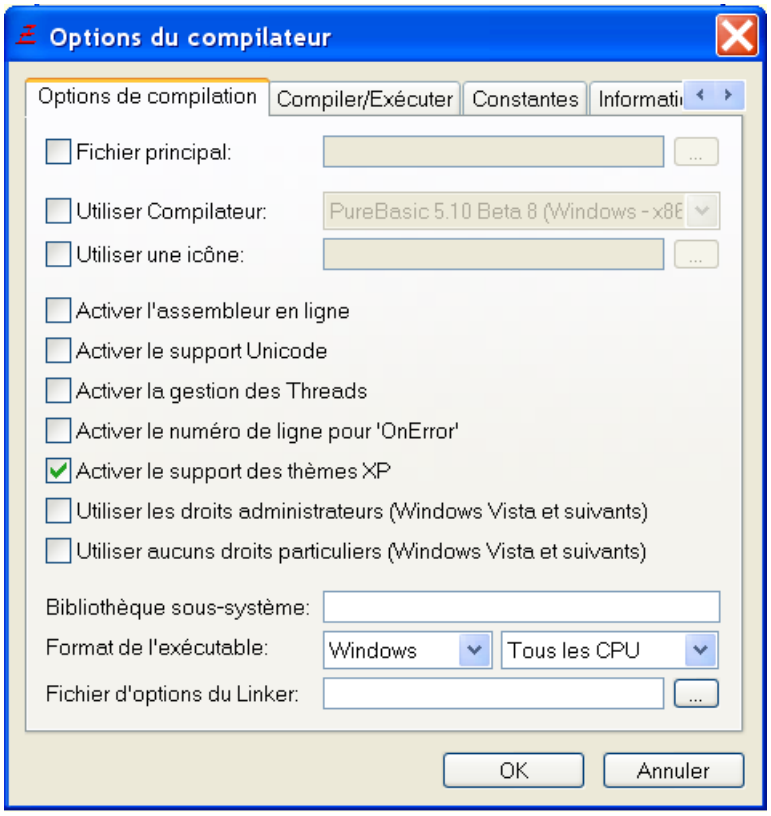
Créer une cible

Quand un projet est ouvert, ce sous-menu montre toutes les cibles disponibles et permet d'en compiler directement une d'entre elles.

Créer toutes les cibles

Quand un projet est ouvert, ce menu compile toutes les cibles qui ont l'option idoine activée dans les options de compilation. Une fenêtre s'ouvre permettant de suivre l'état d'avancement des compilations.

La fenêtre d'options de compilation pour les fichiers hors projets



Fichier principal
En activant cette option, il est possible de définir un fichier principal pour la compilation du fichier actuel. Ce sera ce fichier qui sera alors utilisé lors de la compilation et non le fichier en cours d'édition. Note : quand cette option est utilisée, le fichier en cours d'édition doit être enregistré avant de lancer la compilation, sinon les modifications ne seront pas prises en compte (car les fichiers inclus sont chargés à partir du disque). La plupart des options de compilation seront héritées du fichier principal, seuls les paramètres concernant le débogueur et la ligne de commande seront propres au fichier actuel.
Utiliser Compilateur
Cette option permet de sélectionner un compilateur différent de la version actuelle de PureBasic. Cela facilite la compilation d'un programme sous différentes achitectures (x86 ou x64) sans avoir à redémarrer une nouvelle instance de l'IDE. Les compilateurs additionnels sont paramétrables dans les préférences . Si la version du compilateur est identique à celle de l'IDE mais que le processeur géré est différent, le débogueur intégré peut quand même être utilisé pour déboguer cet exécutable. Par exemple, une exécutable compilé avec la version x86 du compilateur peut être débogué avec l'IDE x64 et vice versa. Si la version du compilateur et de l'IDE ne correspondent pas, alors le débogueur indépendant sera utilisé pour éviter des problèmes potentiels.
Utiliser une icône (Uniquement pour Windows et MacOS X)
Il est possible de spécifier une icône qui sera associée à l'exécutable et affichée dans l'Explorateur Windows (ainsi que dans le coin supérieur gauche des fenêtres du programme et dans la barre des tâches).
Windows : L'icône doit être au format ICO (icônes Windows).
MacOS X : Le fichier icône doit être au format ICNS (icônes Macintosh). Pour créer un tel fichier, il faut 4 fichiers PNG de dimensions 128x128, 48x48, 32x32 et 16x16 de l'image qui sera utilisée pour l'icône. Ensuite l'outil "Icon Composer" fourni avec les outils de développement Apple servira à créer le fichier final (il devrait se trouver dans le dossier : /Developer/Applications/Utilities/). Il sera peut-être nécessaire de redémarrer l'explorateur (Finder) pour l'afficher.
Activer la colorisation des mots clés assembleur
Active la colorisation des mots clés de l'assembleur. Voir la section Assembleur en ligne x86 pour plus d'informations.
Activer la gestion des Thread
Informe le compilateur qu'il doit utiliser les routines multi-threadées lors de la création de l'exécutable

(voir la bibliothèque Thread pour plus d'information).
Ce mode permet aussi au débogueur d'afficher correctement les informations si des threads sont utilisés. Sans cette option, le débogueur peut afficher un numéro de ligne erroné si une erreur intervient à l'intérieur d'un thread.
Activer le numéro de ligne pour OnError (Uniquement pour Windows)
Ajoute des informations dans l'exécutable pour pouvoir identifier la ligne qui a provoqué une erreur (à utiliser en combinaison avec les commandes de la bibliothèque OnError). Cette option influe légèrement sur les performances du programme.
Activer le support des thèmes (Uniquement pour Windows)
Ajoute un fichier permettant la gestion des thèmes Windows (fenêtres et gadgets skinées), lorsque le programme est exécuté sur Windows XP, Windows Vista, Windows 7, Windows 8 (fortement recommandé) ou Windows 10.
L'application nécessite les droits administrateurs (Windows Vista et suivants) (Windows seulement)
L'exécutable créé sera toujours lancé avec les droits d'administration (i.e vous devez être logué en tant qu'administrateur système) sous Windows Vista et suivants. (il ne s'exécutera pas si le mot de passe de l'administrateur n'est pas donné). Cette option sera définie pour des programmes qui ont besoin d'un accès à des dossiers ou à des zones de registres à accès restreint afin d'obtenir un accès complet.
Si cette option est activée, le débogueur indépendant sera automatiquement sélectionné pendant la phase de débogage, ainsi le programme pourra être testé sous le mode administrateur.
Note : cette option n'a aucun effet quand le programme est lancé sous d'autres versions de windows.
L'application ne nécessite aucun droits particuliers (Windows Vista et suivants) (Windows seulement)
Cette option désactive la «virtualisation» de cet exécutable sur Windows Vista et suivants. La Virtualisation provoque la redirection des fichiers et les accès au Registre vers un dossier utilisateur si l'utilisateur n'a pas les droits nécessaires à l'opération. (ce qui permet de conserver la compatibilité avec d'anciens programmes)
Notez que cette redirection se fait sans en aviser l'utilisateur ce qui peut conduire à une certaine confusion si il essaye de trouver les fichiers sauvegardés sur le système de fichiers. Pour cette raison, il est recommandé de désactiver cette fonctionnalité si le logiciel est conforme avec les règles d'accès aux fichiers ou du registre de Vista.
Note : cette option n'a aucun effet quand le programme est lancé sous d'autres versions de windows. Elle ne peut pas être combinée avec l'option "Mode Administrateur" ci-dessus.
bibliothèque Sous-système
Il est possible d'utiliser différents sous-systèmes lors de la compilation. Plus d'un sous-système peut être spécifié, séparés par un espace. Pour plus d'informations, consultez sous-systèmes .
Format de l'exécutable
Permet de déterminer le type de l'exécutable qui sera généré :
Windows : format par défaut sous Windows, convient pour tout type d'applications.
Console : un exécutable avec une console par défaut. Il est toujours possible de créer des fenêtres, boutons etc. avec ce type d'exécutable, mais une console sera toujours ouverte quand le programme est exécuté. Si le programme est exécuté à partir d'une ligne de commande, il utilisera ce terminal pour les sorties textes, là où un programme du type "Windows" nécessite la création d'une nouvelle console (à l'aide d'OpenConsole()). Si un programme peut avoir son entrée ou sa sortie redirigée (au travers de 'pipes'), cette option doit être utilisée.
DLL Partagé : crée une DLL Windows. Voir "Créer une DLL" pour plus d'informations.
Note : Quand un code source de type DLL est lancé avec "Compiler/Exécuter", il sera traité comme un programme classique, pour permettre de tester facilement les fonctions de la DLL. La DLL sera réellement créée en appellant la commande "Créer un exécutable".
Optimisations CPU (à coté de "Format de l'exécutable")
Cette option permet d'inclure des fonctions spécialement optimisées pour un type de processeur, lorsqu'elles sont disponibles.
Tout CPU : Les fonctions génériques qui fonctionnent sur tous les processeurs seront utilisées.
CPU Dynamique : Les fonctions génériques ainsi que les fonctions spécifiques à tous les CPU supportés seront intégrées à l'exécutable. Le choix des fonctions à utiliser se fera au démarrage de l'exécutable, en détectant dynamiquement le processeur sur lequel il est lancé. Cela crée des exécutables plus gros, mais qui seront aussi rapides que possible.
Les autres options : Les fonctions spécifiques à un CPU seront utilisées (ainsi que les fonctions génériques pour celle qui n'ont pas de version spécifique). L'exécutable ne fonctionnera que sur le processeur choisi.

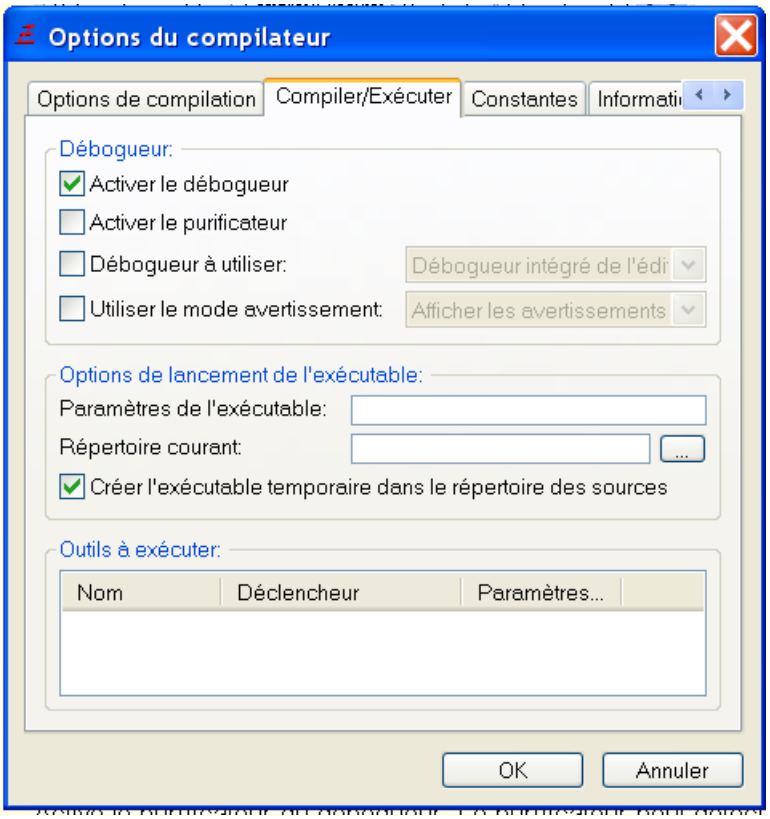
Note : Pour le moment, aucune fonction de PureBasic n'intègre de version spécifique, mais quelques bibliothèques utilisateurs en proposent.

Fichier d'options du Linker

Un fichier texte peut être spécifié ici avec de nouvelles options de ligne de commande à transmettre à l'éditeur de liens lors de la création de l'exécutable. Le fichier doit contenir une option par ligne.

Compiler/Exécuter

Cette section contient des options qui affectent la façon dont l'exécutable est exécuté à partir de l'IDE pendant les tests. Sauf pour l'option 'outils', ils n'ont aucun effet quand le menu "Créer un exécutable" est utilisé.



Activer le Débogueur

Définit l'état du débogueur (on/off) pour le code source en cours, ou si l'option du fichier principal est utilisé, pour ce fichier aussi. On peut utiliser directement le menu du débogueur

Activer le Purifier

Active le purificateur du débogueur. Le purificateur peut détecter certains types d'erreurs de programmation comme écrire au delà des capacités d'un tampon en mémoire. Voir outils de débogage interne pour plus de détails.

Débogueur à utiliser

Permet de choisir un type de débogueur différent pour le fichier en cours uniquement. Si cette option est désactivée, le débogueur par défaut est utilisé, c'est celui spécifié dans les préférences .

Utiliser le mode avertissement

Permet d'ignorer, d'afficher ou de traiter les avertissements.

Paramètre de l'éxecutable

La chaîne donnée ici sera passée en tant que ligne de commande pour le programme lorsqu'il est exécuté à partir de l'IDE.

Répertoire courant

Le répertoire spécifié ici sera défini comme le répertoire courant pour le programme lorsqu'il est exécuté à partir de l'IDE.

Créer l'exécutable temporaire dans le répertoire source

Avec cette option activée, le fichier exécutable temporaire sera placé à l'intérieur du répertoire source.

Cela peut être utile si le programme dépend de fichiers à l'intérieur du répertoire source. Avec cette option désactivée, l'exécutable est créé dans le répertoire temporaire de système.

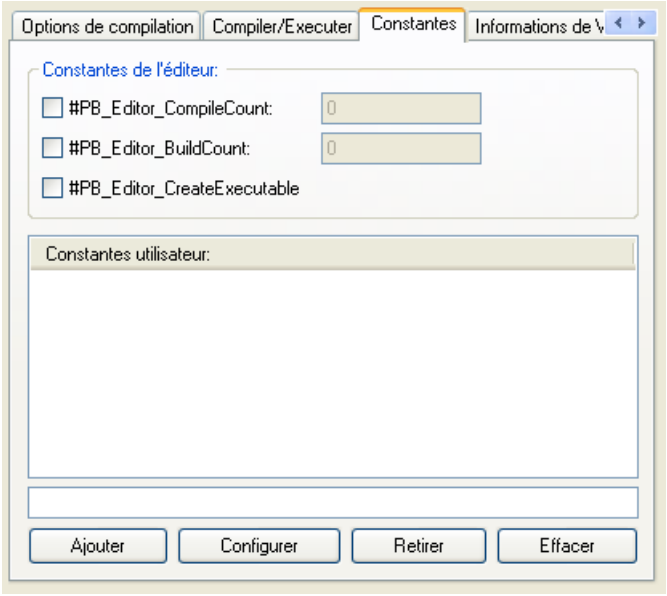
Outils à exécuter

Des outils externes peuvent être activés. La colonne "Paramètres globaux" indique si l'outil est activé ou désactivé dans la configuration des outils . Un outil ne sera exécuté pour la source que s'il est à la fois activé globalement et pour la source en cours.

Remarque : Pour qu'un outil puisse être énuméré ici, il doit avoir l'option "Activer l'outil sur une base per-source" dans la configuration des outils et être exécuté par un déclencheur qui est associé à un fichier source. (c.-à-pas par le menu de démarrage ou de l'éditeur par exemple)

Constantes

Dans cette section, un ensemble de constantes spéciales pour l'éditeur aussi bien que des constantes personnalisées peuvent être définies pour ce qui sera prédéfini au moment de la compilation du code source.



#PB_Editor_CompileCount

Si cette constante est activée, elle contiendra le nombre de fois que le code a été compilé en choisissant aussi bien "Compiler/Exécuter" que "Créer un exécutable" dans le menu. Le compteur peut être édité manuellement dans le champ de saisie.

#PB_Editor_BuildCount

Si cette constante est activée, elle contiendra le nombre de fois que le code a été compilé en choisissant "Créer un exécutable" dans le menu. Le compteur peut être édité manuellement dans le champ de saisie.

#PB_Editor_CreateExecutable

Si cette constante est activée, elle aura la valeur 1 si le code est compilé en choisissant "Créer un exécutable" dans le menu ou 0 si "Compiler/Exécuter" est utilisé.

Constantes utilisateurs

Les constantes personnalisées peuvent être définies et facilement activé / désactivé avec les cases à cocher. Les constantes devraient être ajoutées à mesure qu'elles apparaissent dans le code source. Cela fournit un moyen pour activer / désactiver certaines fonctionnalités dans un programme et on peut activer / désactiver ces fonctionnalités avec CompilerIf/CompilerEndIf .

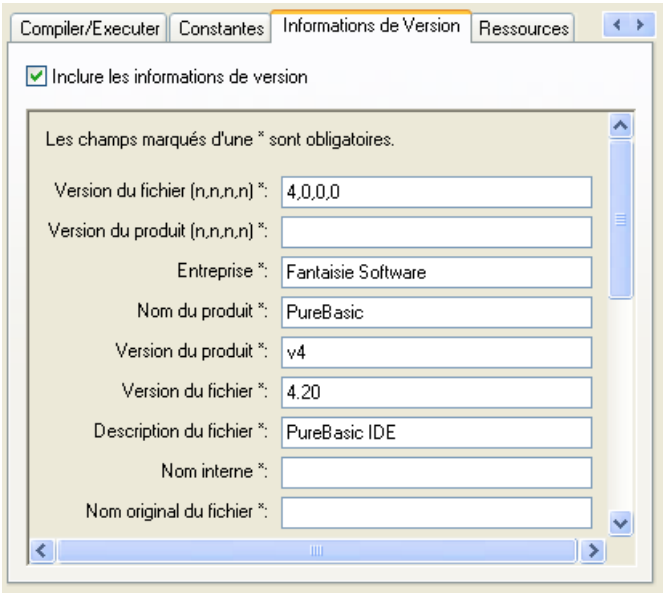
Dans la définition de ces constantes, les variables d'environnement peuvent être utilisées dans un style "bash" avec un "$" devant. La variable d'environnement sera remplacée par la définition avant de compiler le source.

Exemple : #Creator=$USERNAME

Ici,le $USERNAME sera remplacé par le nom de l'utilisateur connecté sur les systèmes Windows. Si une variable d'environnement n'existe pas, elle sera remplacée par une chaîne vide.

Note : Pour tester dans le code source si une constante est définie ou non, la fonction Defined() dans fonction du compilateur peut être utilisée.

Informations de version

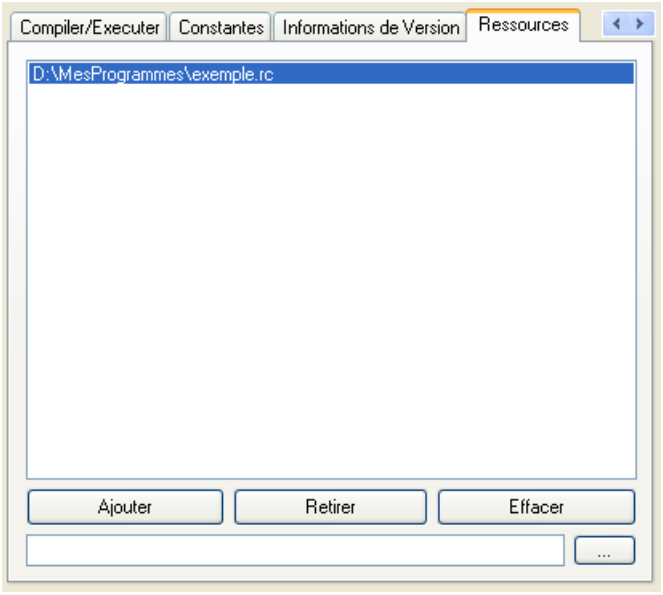


Pour Windows seulement : Si cette fonction est activée, une 'ressource' sera ajoutée à l'exécutable final contenant les informations de version du programme. Il est possible de les consulter en utilisant l'explorateur Windows et de sélectionner 'Propriétés' dans le menu contextuel. D'autres outils peuvent utiliser ces informations, comme par exemple les installateurs de programmes.
Les champs marqués avec une '*' sont indispensables, sinon les informations ne seront pas affichées correctement sur toutes les versions de Windows.
Les deux premiers champs doivent contenir 4 nombres séparés par des virgules. Tous les autres champs peuvent contenir n'importe quel texte. Dans les 3 boîtes vides, il est possible de définir ses propres champs à inclure dans les informations de version.
Dans tous les champs texte, il est possible d'intégrer des tags spéciaux qui seront remplacés par leur valeur associée lors de la compilation :
%OS : remplacé par la version de Windows utilisée pour compiler le programme.
%SOURCE : remplacé par le nom du fichier source (sans son chemin).
%EXECUTABLE : remplacé par le nom de l'exécutable créé (fonctionne uniquement quand "Créer un exécutable" est utilisé).
%COMPILECOUNT : remplacé par la valeur de la constante #PB_Editor_CompileCount.
%BUILDCOUNT : remplacé par la valeur de la constante #PB_Editor_BuildCount.
De plus, il possible d'utiliser n'importe quels tags supportés par la commande FormatDate(). Ils seront alors remplacés par leur valeur par rapport à la date de la compilation (ex : %yy correspondra à l'année de la compilation).
Définition des 3 champs du bas :
OS du fichier
Spécifie l'OS pour lequel ce programme est compilé (utiliser VOS_DOS ou VOS_WINDOWS16 n'a pas vraiment de sens pour un programme PureBasic, ils sont présents uniquement pour avoir une liste exhaustive).
Type du fichier
Définit le type de l'exécutable (ici seuls VFT_UNKNOWN, VFT_APP et VFT_DLL ont un sens pour les programmes PureBasic).
Langue
Spécifie la langue dans laquelle sont décrites ces informations de version.
Les valeurs des champs sont accessibles lors de la compilation du programme de l'IDE en utilisant les constantes suivantes (même ordre) :

```
#PB_Editor_FileVersionNumeric
#PB_Editor_ProductVersionNumeric
#PB_Editor_CompanyName
#PB_Editor_ProductName
#PB_Editor_ProductVersion
```

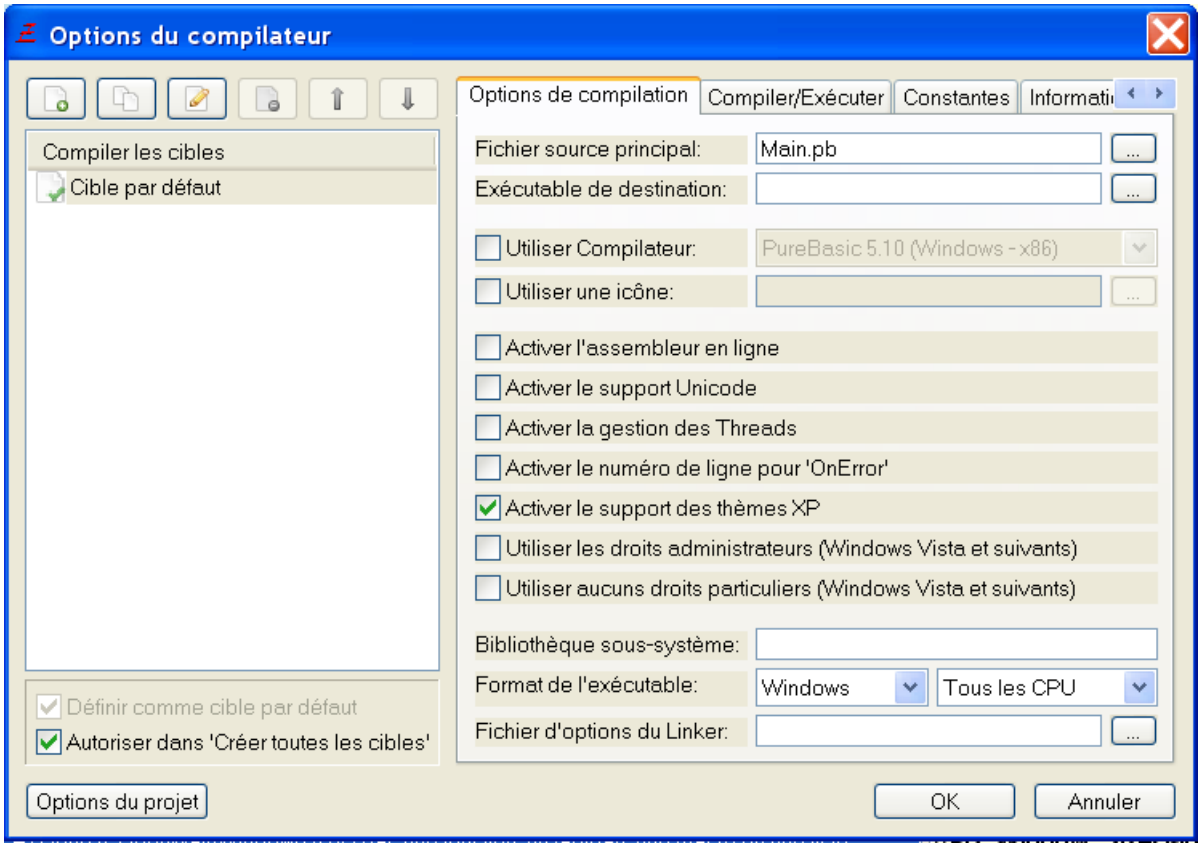
```
#PB_Editor_FileVersion
#PB_Editor_FileDescription
#PB_Editor_InternalName
#PB_Editor_OriginalFilename
#PB_Editor_LegalCopyright
#PB_Editor_LegalTrademarks
#PB_Editor_PrivateBuild
#PB_Editor_SpecialBuild
#PB_Editor_Email
#PB_Editor_Website
```

Ressources



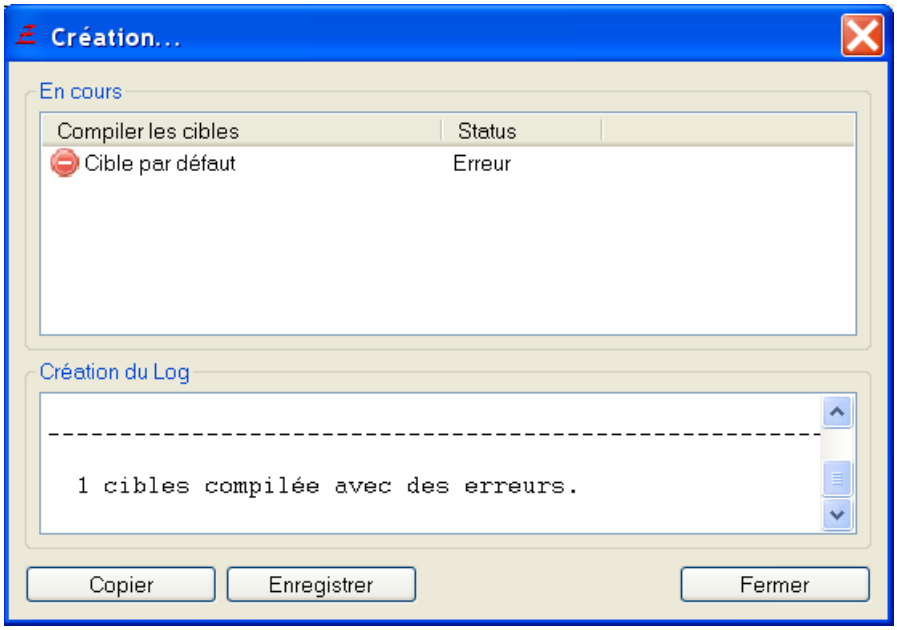
Pour Windows seulement : Permet d'inclure autant de scripts de ressources (fichiers *.rc) que nécessaire. Ils seront alors compilés et ajoutés à l'exécutable, et accessibles dans le programme via les commandes API (étant donné que les ressources sont spécifiques à Windows, PureBasic ne les supporte pas de manière implicite, pour plus de renseignements consultez la MSDN). Pour créer facilement ces scripts, il est conseillé d'utiliser un éditeur de ressources tel que celui présent dans 'PellesC'.

La fenêtre d'options de compilation pour projets



La fenêtre "options de compilation pour projets" permet de définir plusieurs cibles de compilation. Chaque cible est un ensemble de paramètres de compilation avec un fichier source principal et un exécutable. La partie gauche de la fenêtre affiche la liste de toutes les cibles disponibles. La barre d'outils juste au dessus permet de créer, effacer, copier, éditer ou déplacer une cible.
La cible par défaut sera celle qui sera exécutée quand le menu "Compilateur/Exécuter" est sélectionné. Elle peut être rapidement changée par la case à cocher "Définir comme cible par défaut" ou à partir du menu "Compilateur". L'option "Autoriser dans 'Compiler toutes les cibles"' indique si cette cible doit être compilée quand le menu "Compiler toutes les cibles" est utilisé.
La partie droite de la fenêtre est quasiment la même qu'en mode hors projet, et reflète les paramètres de compilation de la cible actuellement selectionnée. Les seules différences sont les champs "Fichier source d'entrée" et "Exécutable de destination" qui doivent être renseignés pour chaque cible, car sans ces informations, il n'est pas possible de compiler la cible automatiquement.
En mode projet, les informations relatives à chaque cible sont enregistrées dans le fichier projet, et pas dans les fichiers sources. Les informations qui sont propres à chaque fichier (comme l'état du pliage) sont toujours enregistrées pour chaque fichier à l'endroit indiqué dans les Preferences .

La fenêtre de compilation des cibles



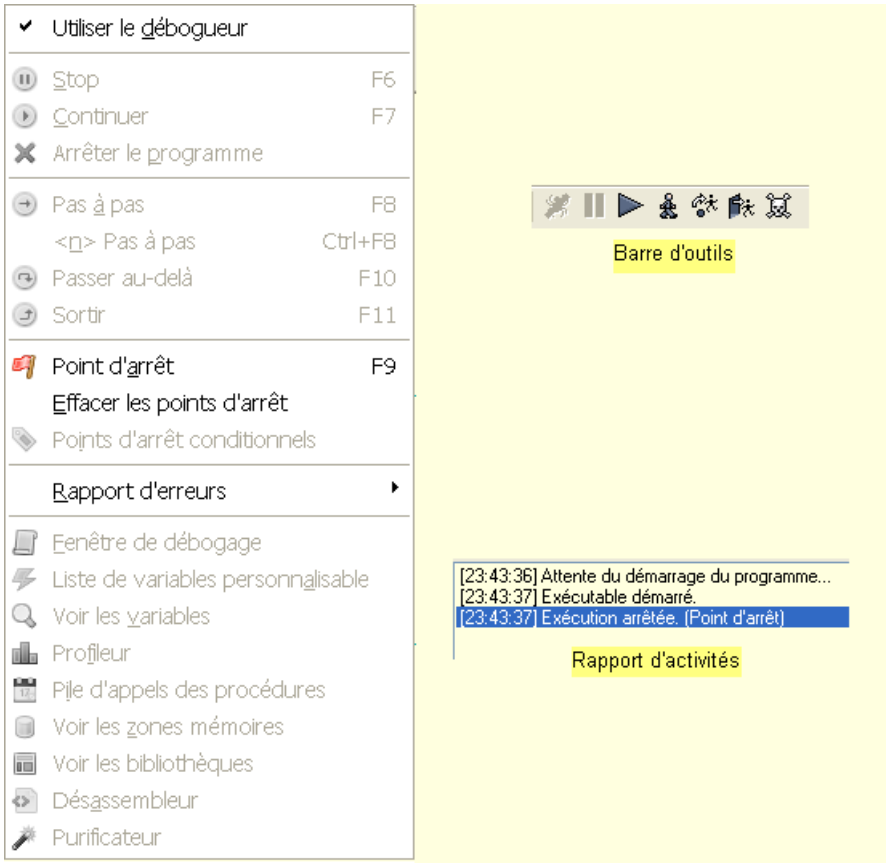
Quand le menu "Créer toutes les cibles" est sélectionné pour le projet en cours, toutes les cibles qui ont l'option idoine activée seront compilées dans l'ordre où elles apparaissent dans la liste des cibles. La fenêtre de progression montre la cible et l'ensemble des cibles ainsi que leurs statuts. Quand le processus de compilation est fini, le log peut être copié dans le presse-papier ou enregistré sur le disque.

Chapitre 13

Utiliser le débogueur

PureBasic est fourni avec un puissant débogueur pour trouver rapidement les erreurs et les bugs des programmes. Il permet de contrôler le flux d'exécution du programme, d'observer le contenu des variables, tableaux et listes, d'afficher des informations de débuggage etc. Il supporte aussi des fonctions avancées tels que l'affichage du contenu des registres (assembleur), voir le contenu de la pile ou d'une zone de mémoire quelconque. De plus il permet de faire du déboguage à distance, via le réseau.
Pour activer le débogueur, il suffit de choisir "Utiliser le débogueur" dans menu "Débogueur" (ou de l'activer dans les "Options de compilation"). En choisissant le menu "Compiler avec le débogueur", le débogueur sera activé seulement pour une compilation.
Le débogueur du PureBasic se présente sous 3 formes :
- Un débogueur intégré directement dans l'IDE, pour une utilisation facile et rapide. C'est ce débogueur qui propose le plus de fonctionnalités.
- Un débogueur indépendant, qui est utile dans plusieurs cas spéciaux (par exemple si le programme est déjà en cours de débuggage et qu'il doit être exécuté une nouvelle fois) ou pour être utilisé par un éditeur de code tierce. Il propose quasiment toutes les fonctionnalités du débogueur intégré, mais parce qu'il est séparé de l'IDE, la rapidité des commandes est légèrement diminuée.
De plus il permet de déboguer un programme à distance, via le réseau.
Un débogueur en ligne de commande uniquement. Le but premier de ce débogueur est de pouvoir tester et développer un programme PureBasic sur un système dépourvu d'environnement graphique (comme un serveur linux), et/ou développer à distance via SSH.
Le type de débogueur à utiliser est sélectionnable dans les préférences.
Bien entendu, quand un programme utilise le débogueur, il fonctionne bien plus lentement. Cela ne devrait pas poser de problèmes majeurs étant donné que la rapidité est toujours acceptable, et que ce mode est utilisé uniquement pendant le développement.
Si quelques parties du programme ont déjà été testées et nécessitent une rapidité maximale même pendant les phases de déboguages, il est possible d'utiliser les directives de compilation DisableDebugger et EnableDebugger autour de ces parties.

Le débogueur intégré



Toutes les commandes relatives au débogueur lorsqu'un programme est en cours d'exécution sont disponibles à partir du menu "Débogueur" (ou de la barre d'outils et des raccourcis claviers).
Tant que le programme est en cours de débuggage, tous les fichiers sources qui sont en rapport avec ce programme sont verrouillés en lecture seule jusqu'à la fin de son exécution. Ceci est fait pour assurer que le code qui sera affiché lors du 'pas à pas' ou lors d'une erreur sera correct (et qu'il n'a pas été édité entre temps sans avoir été recompilé).
A noter qu'un programme ne peut être débogué qu'une seule fois par le débogueur intégré. Si le même programme est de nouveau compilé pour être debuggé, le débogueur indépendant sera utilisé. Par contre il est possible de déboguer plusieurs programmes différents simultanément avec le débogueur intégré.
Astuce
Sous Windows, le menu "Débogueur" est aussi ajouté au menu système de la fenêtre principale de l'IDE (le menu qui apparaît lors d'un click sur l'icône située dans le coin supérieur gauche de la fenêtre). Cela permet aussi d'accéder à ce menu à partir de la barre des tâches, en cliquant avec le bouton de droite de la souris sur l'icône correspondant à l'IDE.

Controle de l'exécution

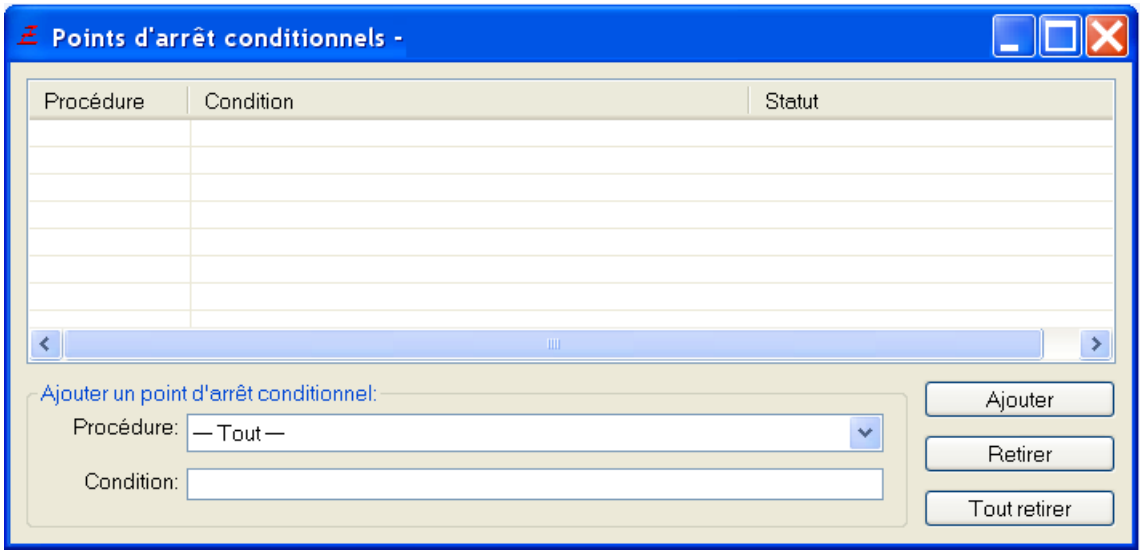
Ces fonctions permettent le contrôle sur le déroulement du programme en cours de débuggage. Le programme peut être stoppé, exécuté pas à pas (ligne par ligne), examiné (voir le contenu des variables à cet instant etc.). Quand le programme est stoppé, la ligne qui va être exécutée est marquée (par défaut en bleu clair) dans le code source correspondant.
L'état du programme est indiqué dans la barre d'état de l'IDE et dans le rapport d'activité.
Commandes du menu permettant le contrôle du programme :
Stop
Stoppe l'exécution du programme et affiche la ligne qui va être exécutée.
Continue
Reprend l'exécution du programme, de manière normale.
Tuer le programme

Force le programme à se terminer et ferme toutes les fenêtres de débuggage associées à ce programme.
Pas
Exécute la ligne du programme actuellement affichée et stoppe de nouveau l'exécution.
Pas <n>
Exécute le nombre de lignes indiqué et stoppe l'exécution du programme.
Passer au-delà
Exécute la ligne du programme actuellement affichée et stoppe de nouveau l'exécution, comme un 'Pas' normal. La différence survient si la ligne contient un ou plusieurs appels à des procédures . Dans ce cas, les procédures seront toutes exécutées, sans arrêt, contrairement au mode 'Pas' qui rentre dans chaque procédure. Ceci permet de passer rapidement sur des procédures qui sont connues comme étant correctes.
Sortir
Exécute le reste du code de la procédure en cours d'exécution et s'arrête à sa sortie. Si la ligne courante n'est pas dans une procédure, un 'Pas' normal sera fait.

Points d'arrêt (ligne)

Les points d'arrêt sont une autre manière de contrôler l'exécution d'un programme. Avec la commande "Point d'arrêt" du menu, la ligne courante sera considérée comme un point d'arrêt (ou retire le point d'arrêt si cette ligne en était un). Quand le programme atteint un point d'arrêt, alors l'exécution est stoppée sur cette ligne (la ligne n'a pas encore été exécutée). A noter que si le point se trouve sur une ligne ou aucun code n'est exécutable (commentaire, ligne vide, structure etc.), le programme s'arrêtera sur la prochaine ligne exécutable rencontrée.
Un fois que l'exécution du programme a été interrompue par un point d'arrêt, il est possible d'utiliser n'importe quelle commande du "Contrôle de l'exécution" pour continuer à déboguer le programme.
Les points d'arrêts peuvent être ajoutés ou retirés de manière dynamique pendant l'édition ou pendant l'exécution du programme. La commande "Effacer les points d'arrêt" permettent d'enlever tous les points d'arrêt du fichier source en cours d'édition.
Note : Pour ajouter/retirer un point d'arrêt à l'aide de la souris, il faut maintenir la touche 'Alt' appuyée pendant le clic sur la bordure qui affiche les points d'arrêts (la colonne de numérotation des lignes n'est pas prise en compte).

Points d'arrêts (conditionnel)



En plus des points d'arrêts classiques, le débogueur permet d'arrêter l'exécution du programme si une condition donnée est remplie. Par exemple, il est possible de mettre une condition sur la valeur d'une variable et d'arrêter le programme quand elle atteint une certaine limite. La condition prend la forme d'une expression PureBasic évaluable en vrai ou faux. Tout ce que l'on peut mettre après un If, y compris les opérateur logiques tels que And, Or ou Not est accepté. La plupart des fonctions des

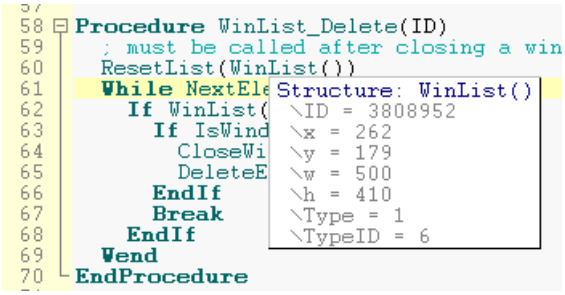
bibliothèques Math , Memory et String ainsi que toutes les fonctions de validation de la forme IsXxx() and the XxxID sont disponibles.
Exemples de conditions :

```
1  MaVariable$ <> "Salut" Or Compteur < 0  ; arrête l'execution si
     'MaVariable$' n'est plus égale à "Salut" ou si 'Compteur' devient
     inférieur à zéro
2  PeekL(*UneAdresse+500) <> 0             ; arrête l'exécution la
     valeur de type long contenu à l'adresse données n'est plus égale à
     zéro
```

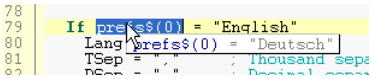
Un point d'arrêt conditionnel peut être ajouté via l'entrée "Point d'arrêt conditionnel" du menu "Debogueur". Il peut se limiter à une procedure particulière, ou il peut être ajouté pour tout le code source. L'entrée "Principal" dans la selection des procédures indique que le point d'arrêt conditionnel devra être évalué seulement en dehors des procédures.
La colonne 'status' affiche le résultat de tous les points d'arrêt conditionnels après leur dernière évaluation. Cela peut être 'vrai', 'faux' ou 'error' si la condition n'est pas une expression valide. Dès qu'une condition est vraie, alors l'exécution du programme s'arrête. Cette condition est automatiquement enlevée de la liste, donc si le programme continue, il ne s'arrêtera pas immédiatement.
Note : La vérification des points d'arrêt conditionnels ralentit l'exécution du programme car les expressions doivent être évaluées à chaque ligne de code. Il vaut mieux les déclarer uniquement quand c'est vraiment nécessaire, pour garder une exécution rapide du programme. Le fait de limiter un point d'arrêt conditionnel à une procédure permet aussi de limiter l'impact sur la rapidité d'exécution, car l'expression n'est évaluée que lorsque la procédure est appelée.

Variables en cours d'exécution

La valeur d'une variable peut être très rapidement vue pendant que le programme est en cours d'exécution en plaçant le curseur de la souris dessus un bref instant dans le code source. Si la variable est actuellement dans la portée et peut être affichée, sa valeur sera affichée dans une info-bulle.



Les expressions plus complexes (par exemple les champs de tableau array) peuvent être consultées en les sélectionnant avec la souris et en plaçant le curseur de la souris sur la sélection.



Des outils du Debogueur offrent également un certain nombre de façons d'examiner le contenu des variables , tableaux ou des listes .

Erreurs dans le programme

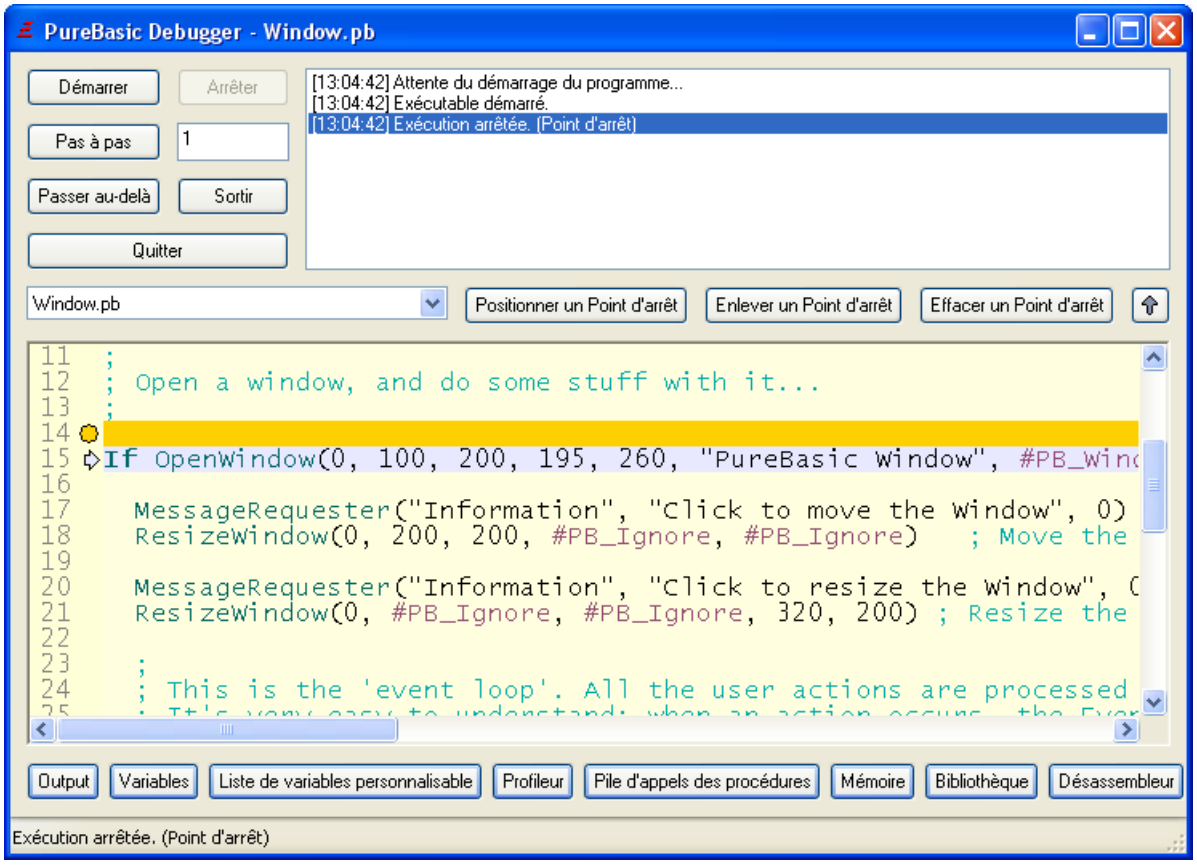
Si le débogueur rencontre une erreur dans le programme en cours de débuggage, il arrêtera l'exécution et marquera la ligne qui contient l'erreur (par défaut en rouge) et affichera le message d'erreur dans la barre d'état et le rapport d'activité.
A ce moment, il est toujours possible de regarder le contenu des variables, de la mémoire et l'historique des appels des procédures, mais les autres fonctionnalités telles que l'affichage des registres ou l'état de

la pile ne sont plus disponibles.
Si l'erreur est considérée comme fatale (comme un accès interdit à une zone mémoire ou une division par 0) il ne sera pas possible de continuer l'exécution du programme. Par contre, si l'erreur est reportée par une commande PureBasic, il est possible de continuer tout de même l'exécution du programme, mais dans ce cas d'autres erreurs anormales peuvent apparaître.
Après une erreur (même fatale), la commande "Tuer le programme" doit etre utilisée pour finir l'exécution du programme et reprendre l'édition du code source. Le programme n'est pas automatiquement tué après une erreur pour permettre au développeur d'utiliser les fonctionnalités du débogueur (comme examiner le contenu des variables) pour essayer de détecter la cause de l'erreur.
Note : il est possible de paramétrer le débogueur pour qu'il termine automatiquement l'exécution du programme en cas d'erreur (voir Paramétrer l'IDE).

Le rapport d'activité

Le rapport d'activité est utilisé pour l'affichage des erreurs de compilation, ainsi que des messages survenant durant le débuggage. Les messages sont toujours affichés dans le rapport d'activité du fichier concerné, donc si une erreur survient dans un fichier inclus, ce dernier sera affiché et un message ajouté dans son rapport d'activité.
Le sous-menu "Rapport d'activité" du menu "Débogueur" contient les commandes pour sa gestion :
Afficher/Cacher
Affiche ou cache le rapport d'activité pour le fichier en cours d'édition.
Effacer
Efface toutes les lignes du rapport d'activité du fichier en cours d'édition.
Copier
Copie le contenu du rapport d'activité dans le presse-papier.
Effacer les erreurs
Une fois que le programme a été tué, les erreurs affichées dans le code sources restent afin de repérer facilement les lignes qui ont causé des problèmes et de les corriger. Cette commande permet d'enlever le surlignage sur les lignes d'erreur. Il est aussi possible de configurer l'IDE pour ôter automatiquement le surlignage des erreurs lorsque le programme se termine (voir Paramétrer l'IDE)

Le débogueur indépendant



Le débogueur indépendant est très similaire à celui intégré à l'IDE et sera seulement expliqué brièvement :
Sur la fenêtre du débogueur sont présents les boutons permettant le contrôle sur l'exécution du programme comme décrit plus haut. Le bouton "Pas" avance d'autant de lignes que défini dans le champ à sa droite. Quand le débogueur est fermé à l'aide du bouton "Quitter" ou en cliquant sur le bouton de fermeture de la fenêtre, le programme en cours de débuggage est immédiatement tué.
La zone de rapport d'activité peut être cachée à l'aide du bouton 'flèche vers le haut' pour rendre la fenêtre de débuggage plus petite.
La zone d'affichage du code source est utilisée pour montrer la ligne en cours d'exécution ainsi que les erreurs ou les points d'arrêt. La liste déroulante située au dessus permet de sélectionner les différents fichiers composant le programme. Les boutons "Mettre point d'arrêt", "Enlever point d'arrêt" et "Effacer points d'arrêt" permettent de gérer dynamiquement les points d'arrêt dans le fichier source affiché. Dans le code, existe également la fonction de survol (du débogueur intégré) pour visualiser rapidement le contenu d'une variable.
Les outils du débogueur peuvent être affichés à l'aides des boutons situés dans la partie inférieure de la fenêtre. L'utilisation de ces outils est la même qu'avec le débogueur intégré.
Note : Le débogueur indépendant n'a pas de configuration spéciale. Il utilise les paramètres du débogueur et de coloration syntaxique de l'IDE. Donc si un éditeur de code tierce est utilisé pour le développement, il convient de régler ces paramètres à l'aide de l'IDE.
Exécution du débogueur indépendant à partir de la ligne de commande :
Pour exécuter un programme compilé en ligne de commande avec le débogueur activé (option /DEBUGGER (Windows) ou -d (Linux/MacOS X)), le débogueur doit être invoqué comme suit :
pbdebugger <fichier exécutable> <paramètres ligne de commande pour l'exécutable>
Si le programme est exécuté immédiatement après une compilation par ligne de commande, le débogueur indépendant est automatiquement utilisé.

Déboguage à distance avec le débogueur indépendant :

La fonctionnalité de déboguage à distance permet d'utiliser l'interface graphique, à la place du débogueur en mode texte, pour déboguer des programmes qui sont exécutés sur des machines distantes, ou dans des machines virtuelles. La compilation du programme doit être gérée séparément, sur la machine distante ou sur la machine locale (en transférant le fichier compilé sur la machine cible).
Les débogueurs de tous les systèmes supportés par le PureBasic sont compatibles à partir du moment où les versions du compilateur utilisé pour créer le programme et du débogueur sont identiques. Par exemple, il est possible de déboguer un programme s'exécutant sur linux x64 avec un débogueur sous Windows x86. Le programme compilé et le débogueur peuvent tous les deux faire office de client ou de serveur pour la connectivité réseau. Un débogueur ne peut déboguer qu'un seul programme. Les connexions multiples ne sont pas possibles.
Lancer le débogueur en mode réseau :
Les paramètres de la ligne commande concernant le réseau :

```
pbdebugger.exe /CONNECT=hote[:port]  [/PASSWORD=motdepasse]
pbdebugger.exe /LISTEN[=interface][:port]  [/PASSWORD=motdepasse]
```

Le mode "connect" permet de lancer le débogueur en mode client, pour qu'il se connecte sur un exécutable qui aura été démarré en mode serveur. Le mode "listen" crée un serveur et attend la connexion d'un exécutable. Si l'adresse IP d'une interface locale est spécifiée, alors le serveur ne sera créé que pour cette interface, sinon le serveur sera créé sur toutes les interfaces disponibles. Si le port n'est pas spécifié, le port par défaut sera utilisé (port 10101).
Le paramètre optionnel 'password' active le chiffrement en AES sur le flux réseau. Si le client est démarré sans l'option 'password' mais que le serveur nécessite un mot de passe, alors il sera demandé d'entrer le mot de passe pour pouvoir continuer.
Lancer l'exécutable en mode réseau :
L'exécutable doit être compilé normalement en mode debug (en utilisant les options du compilateur /DEBUGGER or –debugger). Il pourra alors être démarré à partir de la ligne de commande avec les paramètres suivants :

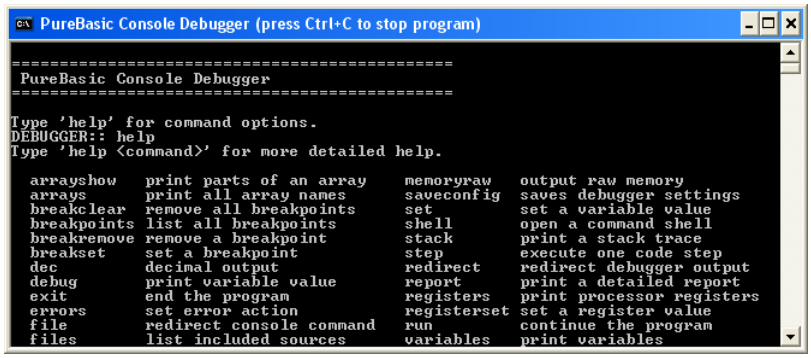
```
yourprogram.exe /DEBUGCONNECT=hote[:port]  [/PASSWORD=motdepasse]
yourprogram.exe /DEBUGLISTEN[=interface][:port]
 [/PASSWORD=motdepasse]
```

Si les options de connexion ne peuvent pas être spécifiées sur la ligne de commande (par exemple parce que le programme lit aussi des paramètres sur la ligne de commande) il existe une alternative en déclarant les variables d'environnement suivantes avant le démarrage du programme (attention à la casse) :

```
PB_DEBUGGER_Communication=NetworkClient;host[:port][;password]
PB_DEBUGGER_Communication=NetworkServer[;interface][:port][;password]
```

Une fois que la connexion réseau est établie entre le débogueur et l'exécutable, la session de déboguage fonctionne de la même manière qu'avec un exécutable local. Si le débogueur est fermé, l'exécutable est fermé aussi. Il n'est pas possible de déconnecter ou de reconnecter le débogueur une fois que la connexion a été établie.

Le débogueur en ligne de commande



Le débogueur en ligne commande ne fait pas partie de l'IDE, il ne sera donc pas expliqué en détail dans cette section.
Lorsque le programme est en cours d'exécution, la combination Ctrl+C dans le terminal permet de le stopper et d'invoquer le débogueur. La commande "help" permet d'avoir un apercu des commandes disponibles et "help <commandname>" affichera une aide sur l'utilisation d'une commande. Sous Windows le débogueur en ligne de commande est utilisé uniquement si l'option /CONSOLE est spécifiée.

Déboguer un programme multi-threadé :

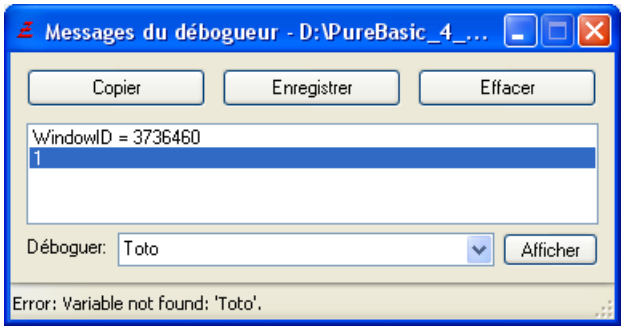
Pour utiliser le débogueur avec un proramme qui utilise des threads , l'option 'Créer un exécutable multi-threadé' doit être activé dans les Options de compilation , sinon les informations affichées par le débogueur concernant les numéros de lignes, les erreurs, les variables locales peuvent être erronées, à cause des threads.
Les limitations suivantes doivent être prises en compte lors du déboguage d'un programme multi-threadé :
Quand le programme est en cours d'exécution, la visionneuse de variables, l'affichage de la pile ou le débogueur assembleur afficheront uniquement les informations du thread principal. Quand un programme est sur arrêt, ils afficheront les informations sur le thread courant. Donc, pour examiner les variables locales d'un thread, l'exécution doit être arrêtée dans ce thread (en utilisant un point d'arrêt ou CallDebugger). Les commandes 'Pas à pas' du débogueur s'appliquent uniquement au thread où l'exécution est arrêtée.
Si une erreur survient, l'exécution est stoppée dans ce thread, donc toute information affichée par la visionneuse de variables ou l'affichage de la pile sera relative à ce thread.
La 'Liste de visualisation' des variables affiche seulement les variables locales du thread principal.
Quand le débogueur s'arrête dans un thread, l'exécution de tous les autres threads est suspendue.

Chapitre 14

Utiliser les outils de débogage

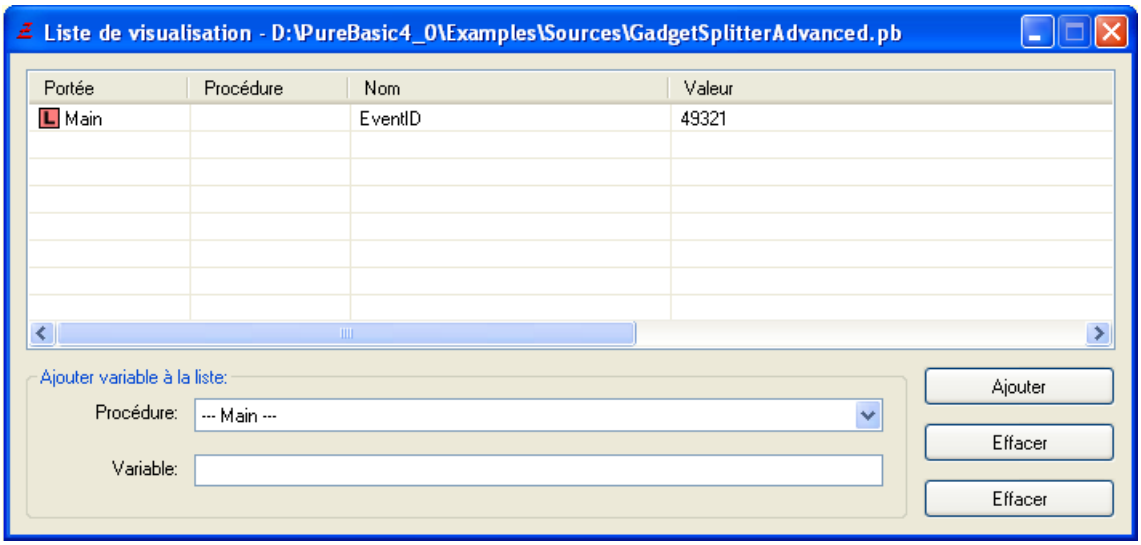
Ces outils proposent de nombreuses fonctionnalités pour inspecter le programme en cours de déboguage. Ils ne peuvent pas être utilisés lorsque que le code source est en train d'être édité. Ils sont tous disponibles à partir du débogueur intégré ou du débogueur indépendant. Le débogueur en ligne de commande propose aussi bon nombre de ces fonctionnalités mais uniquement à l'aide de commandes, rendant leurs utilisations moins facile.
Certains de ces outils permettent d'afficher le contenu des variables. Voici comment est présenté l'affichage pour ces données :
Scope
Le scope d'une variable est la zone dans laquelle elle est valide. Un variable peut être globale , locale , partagée , statique ou threadé , en fonction de sa déclaration dans le code source. Le terme 'byref' ("par référence", c'est à dire en utilisant l'adresse) est utilisé pour indiquer un tableau ou une liste qui a été passé en paramètre d'une procédure.
Type des variables
Le type des variables est indiqué à l'aide d'une icône colorée, pour faciliter leur reconnaissance :
B : Byte
A : Ascii
W : Word
U : Unicode
L : Long
I : Integer
Q : Quad
F : Float
D : Double
S : String (Texte)
Sn : Fixed String (Texte)
Une structure est soit indiquée à l'aide d'un point, ou d'une flèche. Si c'est une flèche, alors le contenu de la structure peut être affiché en double-cliquant sur la ligne. L'icône est alors remplacée par une flèche vers le bas. Une structure représentée par un point ne peut pas être consultée (souvent car il s'agit d'un pointeur vers une structure).
Les tableaux dynamiques à l'intérieur de structures sont représentés avec leurs dimensions. Les listes et les maps à l'intérieur de structures sont représentées avec leur taille et de leur élément courant (s'il existe).

La fenêtre de débug



C'est dans cette fenêtre qu'apparaîtront les résultats de la commande "Debug". Cette commande est un moyen rapide et simple pour afficher des messages destinés au débuggage du programme.
Cette fenêtre sera affichée automatiquement la première fois qu'un message sera émis par le programme. Si elle est fermée, elle ne sera plus affichée automatiquement pour les messages suivants, mais ils seront tout de même enregistrés. Il est possible de copier le contenu de cette fenêtre dans le presse-papier ou de l'enregistrer dans un fichier. Un bouton est aussi présent pour effacer tous les messages précédents.
Le champ en bas de la fenêtre vous permet de saisir une expression qui sera évaluée, et le résultat sera affiché dans la fenêtre du débogueur. Ceci permet de contrôler rapidement la valeur d'une variable ou le champ d'un tableau sans avoir à lancer un des outils de débogage. Pour afficher le résultat de l'expression , appuyez sur la touche [entrée] ou cliquez sur le bouton "Afficher". Si l'expression ne peut pas être évaluée, un message d'erreur est affiché dans la barre d'état.
L'expression peut être n'importe quelle expression PB valide, à l'exception des expressions logiques ou contenant des mots clefs . Elle admet les variables , tableaux , listes , constantes et aussi quelques commandes des bibliothèques Math , Memory et String .

La fenêtre de surveillance



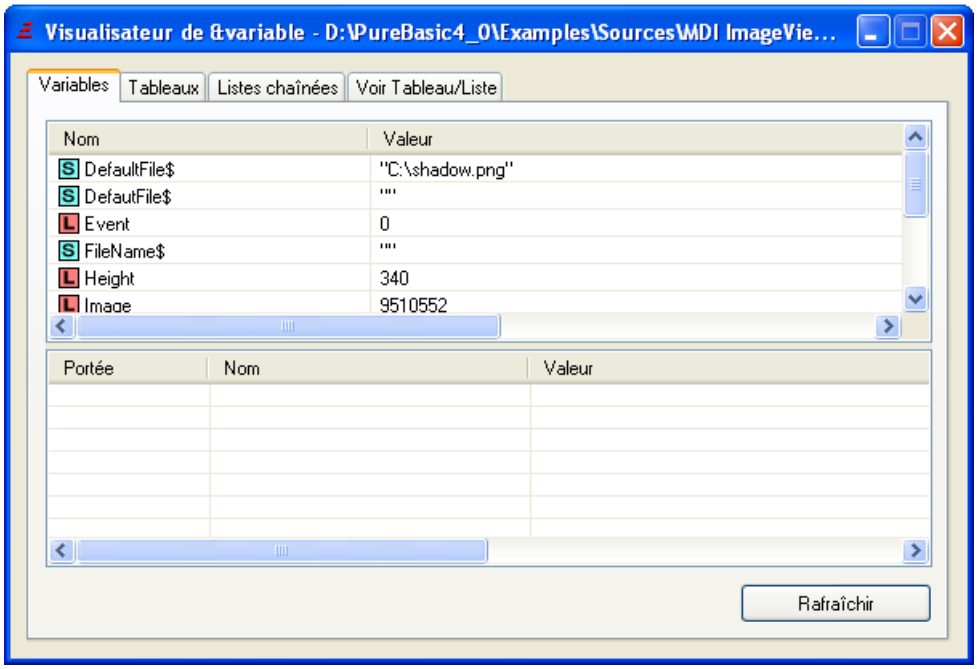
Elle est utilisée pour surveiller en temps réel les valeurs des variables. Il n'est possible d'afficher que des variables de type basique (pas de structures complètes), cependant ces variables peuvent faire partie d'une structure. Les champs de type tableau, liste ou map faisant partie d'une structure ne peuvent pas être affichés dans la liste de surveillance.
Pour ajouter une variable, il faut choisir sa procédure (si c'est une variable locale) ou "— Principal —" si c'est une variable globale, une composante d'un tableau ou d'une liste. Il suffit d'entrer ensuite le nom de la variable dans le champ correspondant et d'appuyer sur "Ajouter".
Exemples :

```
MaVariable$                          - ajoute une variable de type
 string
```

```
Tableau(1, 5)                          ajoute une case d'un tableau
Structure\SousChamp[5]\Valeur          ajoute un champ d'une structure
MaListeChainee()\SousChampStructure    ajoute un champ d'une liste
```

Il est aussi possible d'ajouter une nouvelle variable à la liste de surveillance à partir du "Visualisateur de variables", en cliquant avec le bouton droit de la souris sur une variable et en sélectionnant "Surveiller". Dans la liste, les valeurs des variables surveillées seront affichées. Si la valeur est affichée avec "—", cela veut dire qu'elle n'est pas valide à cet endroit du programme (par exemple une variable locale en dehors de la procédure concernée ou une liste sans élément).
Les variables surveillées sont persistantes entre les sessions de débuggage, et même sauvegardées avec les options de compilation, il n'est donc pas nécessaire de les saisir à chaque fois.

Le visualisateur de variables



Il permet d'examiner les variables, tableaux , listes et maps présents dans le programme. Des les onglets, la partie supérieure montre les éléments globaux and threadés et la partie inférieure les éléments locaux , partagés et statiques .
Le bouton "Actualiser" permet de visualiser les données actuelles du programme en cours d'exécution. Si le programme est stoppé ou en mode pas à pas, les données sont actualisées automatiquement.
Sous Windows, le contenu du visualisateur de variable peut-être trié par nom, scope ou valeur en cliquant sur l'entête de la colonne appropriée.
L'onglet 'Variables'
Cet onglet affiche toutes les variables du programme. En faisant un click-droit sur la variable, il est possible de l'ajouter à la liste de surveillance.
L'onglet 'Tableaux'
Cet onglet affiche tous les tableaux du programme ainsi que leurs dimensions (-1 indique que Dim n'a pas encore été appelé pour ce tableau). En faisant un click-droit sur le tableau, son contenu peut être visualisé dans l'onglet "Voir Tableau/Liste/Map".
L'onglet 'Listes'
Cet onglet affiche toutes les listes du programme, le nombre d'éléments ("-" indique que NewList n'a pas encore été appelé), ainsi que l'index actuel de l'élément courant pour chaque liste ("-" indique qu'il n'y a pas d'élément courant). En faisant un click-droit sur la liste, son contenu peut être visualisé dans l'onglet "Voir Tableau/Liste/Map".
L'onglet 'Maps'
Cet onglet affiche toutes les map du programme, le nombre d'éléments ("-" indique que NewMap n'a pas encore été appelé), ainsi que la clef de l'élément courant pour chaque map ("-" indique qu'il n'y a pas

d'élément courant). En faisant un click-droit sur la map, son contenu peut être visualisé dans l'onglet "Voir Tableau/Liste/Map". **L'onglet 'Voir Tableau/Liste/Map'**
Cet onglet permet d'afficher les éléments d'un tableau, d'une liste ou d'une map, y compris ceux déclarés dans une structure. Pour ce faire, il faut spécifier le nom du tableau, de la liste chaînée ou de la map en incluant les parenthèses "()" à la fin, choisir le type d'éléments à afficher et appuyer sur "Afficher". A noter que les valeurs ne sont pas automatiquement actualisées quand le programme est en mode pas à pas.
"Afficher tout" montre tous les éléments. "Afficher éléments non-nuls" affiche seulement les éléments qui ne sont pas égaux à 0. Cela permet d'afficher des gros tableaux/listes plus facilement si seulement quelques éléments sont utilisés. Une structure est considérée comme "nulle" si tous ses champs sont à 0.
"Afficher partiellement" permet d'afficher seulement une partie du tableau, de la liste ou de la map. Pour les tableaux, l'intervalle peut être spécifié séparément pour chaque dimension en utilisant une virgule. Si une dimension n'est pas du tout spécifiée, tous ses éléments seront affichés.
Voici quelques exemples d'intervalles :

```
"1-2, 2-5, 2" : la première dimension entre 1 et 2, la deuxième entre
 2 et 5, la troisième à 2.
"1, 2-5"      : la première dimension à 1, la deuxième entre 2 et 5
"1, , 5"      : la première dimension à 1, tous les éléments de la
 deuxième dimension, la troisième à 5.
```

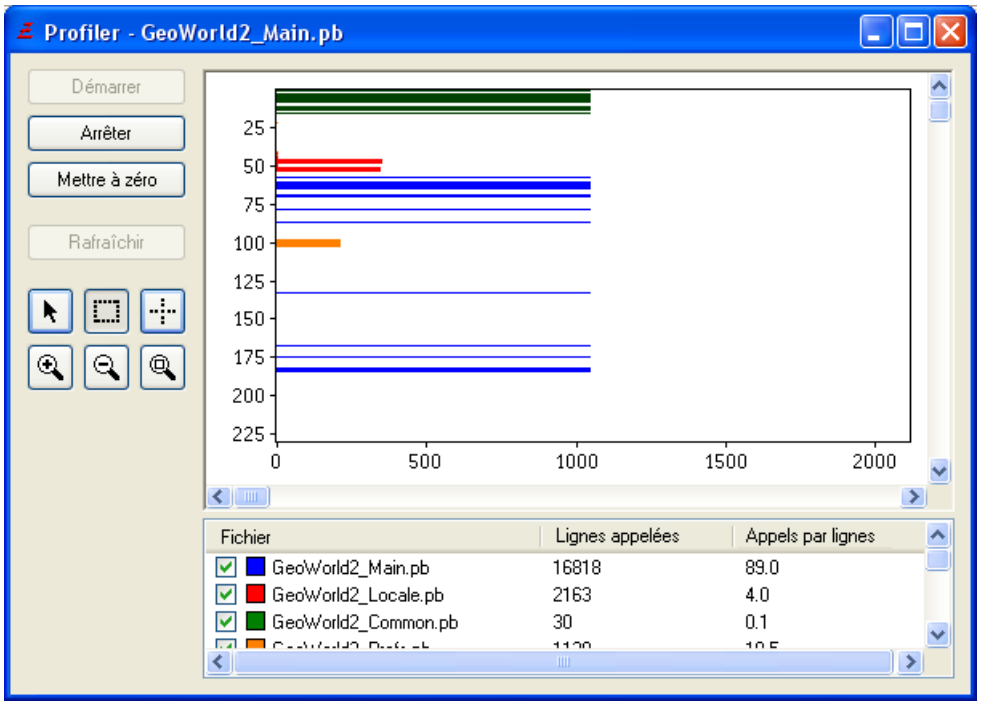
Pour les listes, "Afficher partiellement" accepte une valeur simple ou un intervalle d'éléments (le premier élément a l'index 0).

```
"0"     : le premier élément
"1-3"   : du 2e au 4e élément
```

L'affichage des maps, "Afficher partiellement" est utilisable pour filtrer les clefs des éléments à afficher. Il doit contenir le masque de la clef qui servira de filtre (sans les trémas). Un " ?" représente n'importe quel caractère, un "*" représente n'importe quel nombre de caractères. Voici quelques exemples de masque :

```
"hat" : affichera seulement l'élément avec la clef "hat".
"?at" : affichera tous les éléments de 3 lettres se finissant pas
 "at", comme "bat", "hat" etc.
"h*t" : affichera tous les élémentes qui commencent pas "h" et se
 terminent par "t".
```

Le Profileur



Le profileur est un outil qui permet de compter le nombre d'exécutions de chaque ligne de code. Cela permet d'identifier quelles sont les parties les plus utilisées et donc où les optimisations auront le plus d'effet. Le profileur aide aussi à identifier un problème où une portion du code est exécutée trop souvent à cause d'une erreur.

Enregistrement des données

L'enregistrement est contrôlé par les boutons placés dans la fenêtre du profileur : 'Démarrer', 'Arrêter' et 'Mettre à zéro' (pour tout initialiser). Le bouton 'Rafraîchir' permet de mettre à jour le graphe pendant l'exécution du programme. Les données sont automatiquement mises à jour lorsque le programme est stoppé ou en mode pas à pas. Par défaut le profiler enregistre les données dès le début du programme mais cette option peut-être changée dans les Préférences .

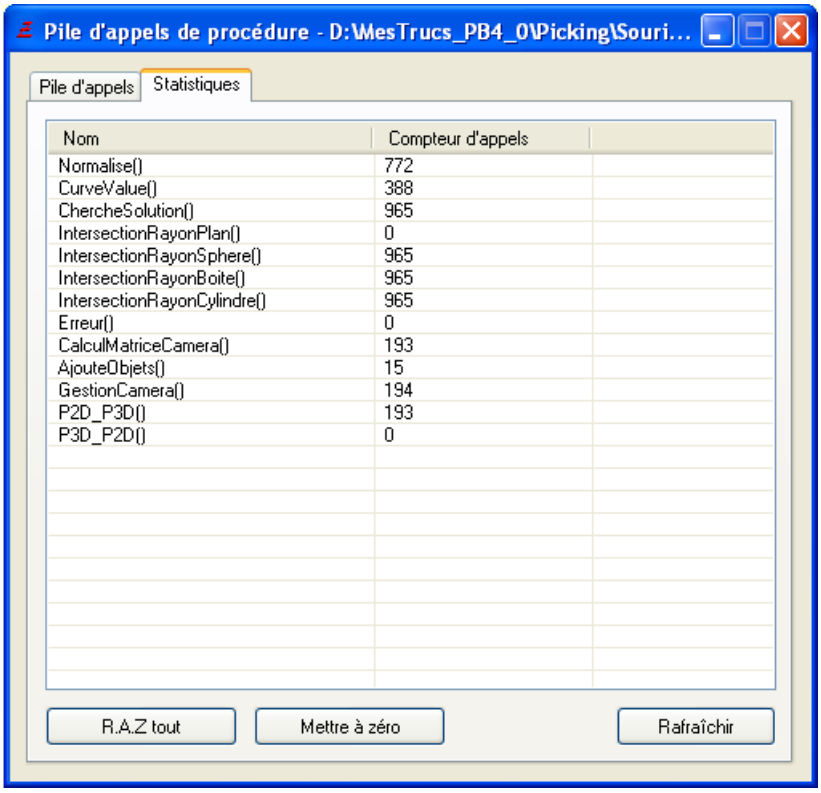
Affichage des données

L'enregistrement est affiché sous forme de graphe, avec comme ordonnées les numéros de lignes et en abscisses le nombre d'exécutions. Si le programme est réparti sur plusieurs fichiers source, la liste des fichiers est présentée en dessous du graphe. Pour afficher un fichier il suffit de le sélectionner ou de cocher la case à cocher, vous pourrez ainsi afficher les résultats de plusieurs fichiers pour mieux les comparer. Un clic droit sur un des fichiers vous permet de changer sa couleur d'affichage dans le graphe.

Utilisation de la souris dans le graphe

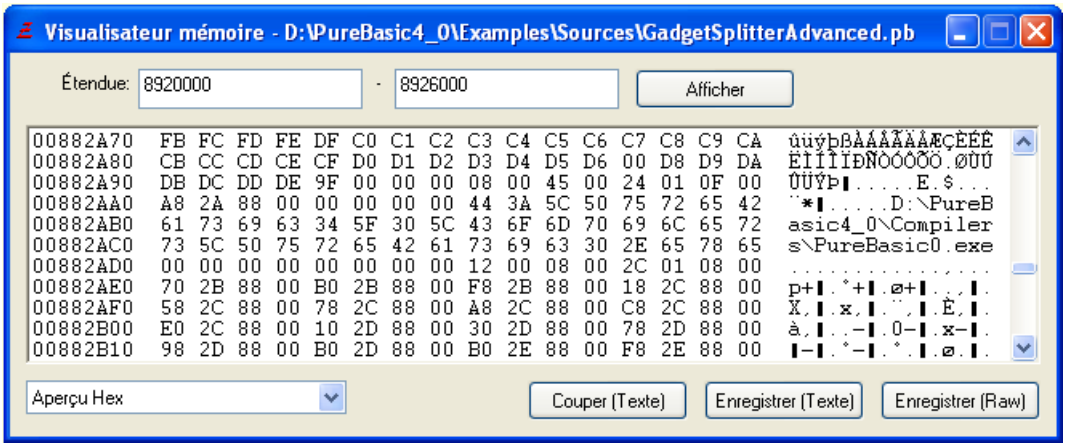
Un clic droit dans le graphe affiche un menu flottant qui autorise à zoomer, montrer la ligne de code sélectionnée dans l'IDE ou le code dans le débogueur. Vous pouvez aussi utiliser les boutons à gauche :
- Flèche : Un clic gauche maintenu permet de faire défiler le graphe.
- Carré : Un clic gauche maintenu permet de sélectionner une zone où faire un zoom.
- Croix : Tant que ce bouton est actif, une croix est affichée et vous aide à lire les numéros de ligne et leur nombre d'appels.
- Zoom : Ces trois boutons permettent de zoomer et d'afficher toutes les lignes.

L'historique des procédures



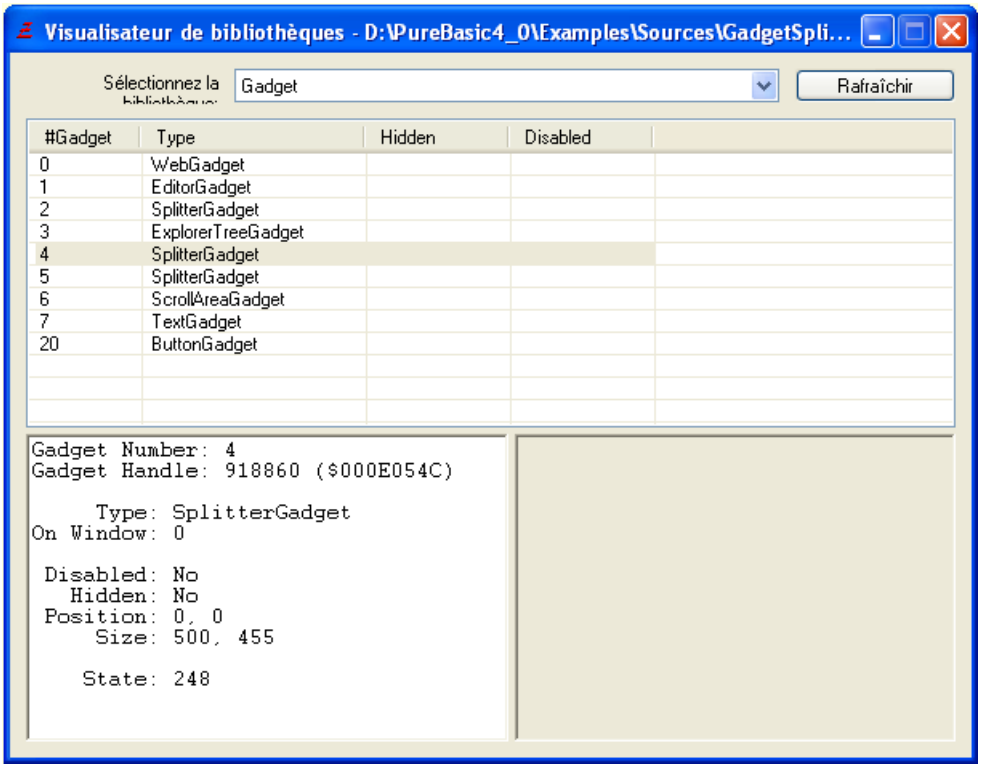
Cet outil montre les procédures qui ont été appelées pour en arriver à la position actuelle du programme. Chaque élément de la liste représente une procédure, la ligne et le fichier où elle est déclarée ainsi que ses arguments qui ont été utilisés lors de son appel. En cliquant sur le bouton "Variables", il est possible de voir les variables de cette procédure.
Cela permet de tracer facilement, à partir de quelle partie du code une procédure a été appelée.
L'historique des procédures ne se met à jour automatiquement que lorsque le programme est arrêté, ou en mode pas à pas. Quand le programme est en cours d'exécution, il est nécessaire d'appuyer sur le bouton "Actualiser" pour mettre à jour la liste.
L'onglet "Statistiques" montre le nombre de fois qu'une procédure a été appelée. Il est possible de réinitialiser un compteur en sélectionnant la procédure puis en appuyant sur "Réinitialiser". De même, en cliquant sur "Réinitialiser tout", tous les compteurs seront remis à 0. Comme pour l'historique des procédures, la mise à jour des compteurs n'est pas automatique quand le programme n'est pas arrêté ou en mode pas à pas, il convient d'utiliser le bouton "Actualiser".

Le visualisateur de mémoire



Il permet d'afficher le contenu d'une zone de mémoire arbitraire de votre programme. Une fois les limites inférieures et supérieures renseignées, cliquez sur "Afficher". (vous pouvez utiliser une valeur décimale, un nombre hexadécimal précédé par le caractère '$' ou n'importe quelle autre expression valide, incluant une variable ou un pointeur provenant du code). Si le contenu du second champ commence par le signe '+', alors il est considéré comme relatif au premier champ. Si la zone de mémoire est valide, elle sera affichée. Si la zone complète ou seulement une portion est invalide, alors un message d'erreur sera affiché. Exemple : "*Buffer + 10" to "+30" affichera les 30 octets dans la mémoire en commençant 10 octets après l'adresse pointée par *Buffer.
La façon de présenter le contenu de la mémoire peut être défini grâce à la liste déroulante située en bas à gauche. Les modes suivants sont disponibles :
Hexadécimal
La mémoire sera affichée en hexadécimal à la manière des visualisateurs hexadécimaux classiques, avec l'adresse de la mémoire à gauche, suivie par le contenu en hexadécimal et la représentation sous forme de caractères dans la colonne de droite.
Tableau Byte/Char/Word/Long/Quad/Float/Double
La mémoire sera montrée sous forme de tableau en fonction du type choisi. Il est possible de configurer ce tableau en mode colonne simple ou multi-colonne dans les préférences (voir Configurer l'IDE)
Texte
Affiche la zone mémoire sous forme de texte, avec tous les caractères de contrôles affichés entre [] (par exemple : "[NULL]" pour le caractère 0). Un retour à la ligne est ajouté après les caractères 'nouvelle ligne' ([LF],[CR]) et [NULL] pour améliorer la lisibilité de la sortie. La zone mémoire peut être interprétée comme une chaîne Ascii, Unicode ou Utf8.
Il est aussi possible d'exporter la zone de mémoire affichée :
Copier (Texte) : Copie la zone de mémoire affichée dans le presse-papier.
Enregistrer (Texte) : Enregistre la zone de mémoire affichée dans un fichier.
Enregistrer (Binaire) : Enregistre la zone de mémoire affichée dans un fichier sous forme binaire.

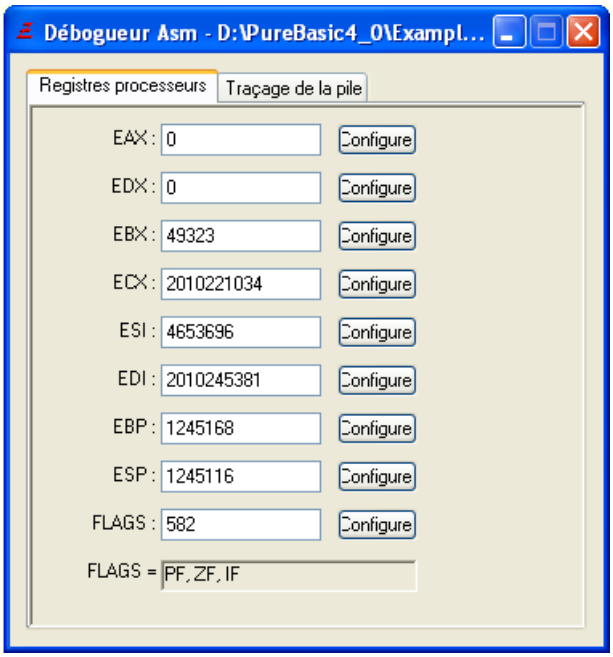
Le visualisateur de bibliothèque



Le visualisateur de bibliothèque donne des informations à propos des objets PureBasic qui ont été créés avec les bibliothèques qui supportent cette fonctionnalité. Par exemple, elle permet de vérifier rapidement les images actuellement chargées dans le programme, ou quels gadgets ont été créés.
Une fois que le programme est démarré, la liste déroulante en haut de la fenêtre peut être utilisée pour

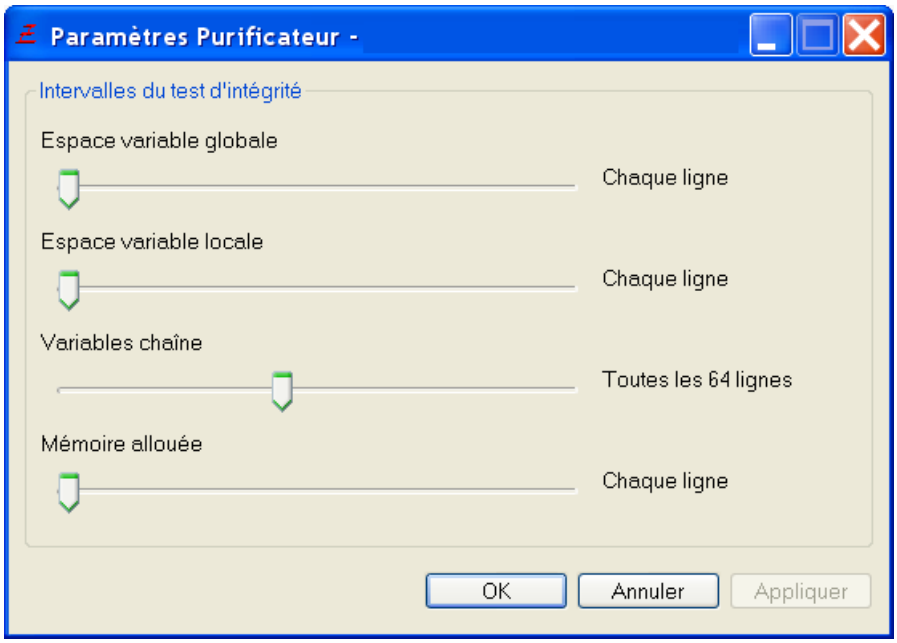
sélectionner la bibliothèque à examiner. La liste de tous les objets de cette bibliothèque apparaîtra et en cliquant sur un objet, des informations relatives à cet objet seront affichées. Certaines bibliothèques affichent même un aperçu de l'objet (sprite, image par exemple).
Si la liste déroulante affiche "Aucune information", cela veut dire que l'exécutable n'a utilisé aucune bibliothèque qui utilise cette fonctionnalité.
Pour l'instant, les bibliothèques suivantes sont supportées :
Thread
Gadget
Window
File
Image
Sprite
XML

Le débogueur assembleur



Le débogueur assembleur est utile pour les développeurs expérimentés qui veulent pouvoir examiner le contenu des registres CPU ou de la pile (surtout quand de l'assembleur en ligne est utilisé).
La fenêtre des registres est uniquement disponible quand le programme est stoppé, ou en pas à pas. Il est possible de changer le contenu des registres en modifiant les valeurs des champs puis en cliquant sur "Changer".
L'analyse de la pile affiche le contenu de la pile du programme par rapport au registre ESP. Si la position actuelle de la pile n'est pas alignée sur une valeur multiple de 4, il n'est pas possible de la décoder correctement, donc elle sera affichée sous forme hexadécimale.
Si le pointeur est correctement aligné, le contenu de la pile est affiché avec des commentaires par rapport aux valeurs rencontrées (détails des valeurs passées en paramètres des fonctions PureBasic etc.)
L'affichage du contenu de la pile est actualisé automatiquement quand le programme est stoppé ou en mode pas à pas. Il est possible de désactiver cette mise à jour automatisée dans les préférences de l'IDE, dans ce cas un bouton "Actualiser" sera disponible.
Note : Le débogueur assembleur n'est pas pour l'instant disponible sur MacOS X.

Le Purificateur



Le purificateur permet de détecter des erreurs d'écriture dans des zones de mémoire interdites (dépassement de tampon). La plupart de ces erreurs entrainent des crashs ou des comportements inattendus, souvent très difficiles et laborieux à localiser, car ils sont aléatoires.
Pour fonctionner, le purificateur a besoin de données particulières générées par le compilateur, c'est pourquoi il est nécessaire de cocher la case "Activer le purificateur" dans les options de compilation .
Le purificateur détecte les erreurs d'écriture grâce à des valeurs uniques autour des variables locales et globales, des chaînes de caractères et des blocs de mémoire alloués dynamiquement. Ces valeurs uniques sont vérifiées régulièrement, et si une de ces valeurs a changé, alors il y a eu une écriture interdite et une erreur est affichée. Ces vérifications ralentissent considérablement l'exécution du programme, particulièrement pour les gros programmes, c'est pourquoi il est possible de régler l'intervalle des vérifications dans la fenêtre du purificateur :

Variables globales
Définit l'intervalle (en nombre de lignes exécutées) pour la vérification de l'intégrité des variables globales.

Variables locales
Définit l'intervalle (en nombre de lignes exécutées) pour la vérification de l'intégrité des variables locales.

Chaînes de caractères
Définit l'intervalle (en nombre de lignes exécutées) pour la vérification de l'intégrité des chaînes de caractères.

Blocs de données dynamiques
Définit l'intervalle (en nombre de lignes exécutées) pour la vérification de l'intégrité des blocs de données alloués dynamiquement avec AllocateMemory() .

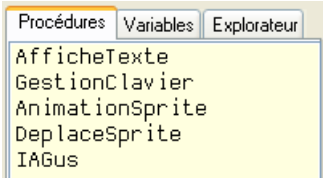
Chapitre 15

Les outils intégrés

L'IDE PureBasic intègre un grand nombre d'outils intégrés, pour rendre la programmation aisée et productive. La plupart d'entre eux peuvent être affichés dans une fenêtre séparée (accessibles alors par le menu) ou dans la palette d'outils située sur le côté de la zone d'édition.
Pour plus d'informations quant à la configuration de ces outils et comment ils sont affichés, voir Configurer l'IDE .

Outils disponibles pour la palette.

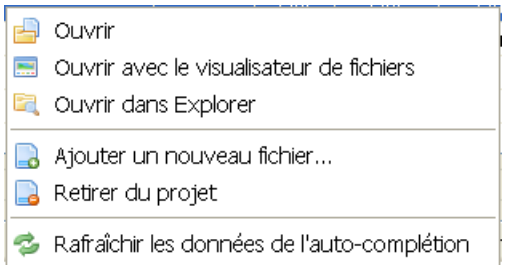
Navigateur de procédures



Cet outil affiche la liste de toutes les procédures déclarées dans le fichier source en cours d'édition. En cliquant sur un élément de cette liste, le curseur changera immédiatement pour aller à la déclaration de cette procédure.
Les macros seront identifiées avec un signe "+" avant le nom.
Il est aussi possible de mettre des commentaires particuliers dans le code qui sera alors aussi affiché dans le navigateur de procédures. Ils ont la forme suivante : ";- <description>". Le ';' démarre le commentaire et le '-' qui le suit immédiatement définit ce type de commentaire. La 'description' sera alors affichée dans la liste et un clic sur cet élément changera la position du curseur pour cette ligne. Ce type de commentaire se distingue dans la liste par l'ajout du caractère '>' devant la description.
La liste des procédures peut être triée, et peut aussi afficher les paramètres de la procédure/macro. Pour ces options, voir Configuration de l'IDE .
Projet

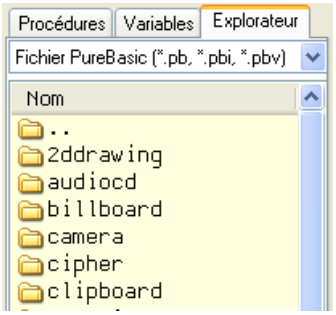
L'outil projet affiche un arbre de tous les fichiers du projet actuellement chargés. Un double-clic sur un

fichier l'ouvrira dans l'IDE. Cela permet un accès rapide à tous les fichiers du projet. Un clic-droit sur un fichier ouvre un menu contextuel qui propose davantage d'options :



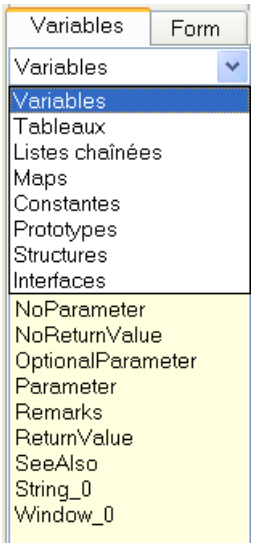
Ouvrir - Ouvre le fichier dans l'IDE.
Ouvrir avec le visualisateur de fichiers - Ouvre le fichier dans le visualisateur intégré de l'IDE.
Ouvrir avec l'explorateur - Ouvre le fichier dans l'explorateur du système d'exploitation.
Ajouter un nouveau fichier - Ajoute un nouveau fichier au projet.
Retirer du projet - Retire le(s) fichier(s) sélectionné(s) du projet.
Rafraîchir les données de l'auto-complétion - Rescanne tous les fichiers du projet pour actualiser les données de l'auto-complétion.

Explorer



L'outil Explorer affiche une liste de fichiers et de répertoires à partir de laquelle il est possible d'ouvrir rapidement n'importe quel type de fichier, en double-cliquant dessus. Les fichiers PureBasic (*.pb, *.pbi, *.pbp, *.pbf) seront chargés directement dans la zone d'édition et les fichiers reconnus par l'éditeur (textes et binaires) seront ouverts par le visualisateur interne de fichiers.

Visualisateur de variables

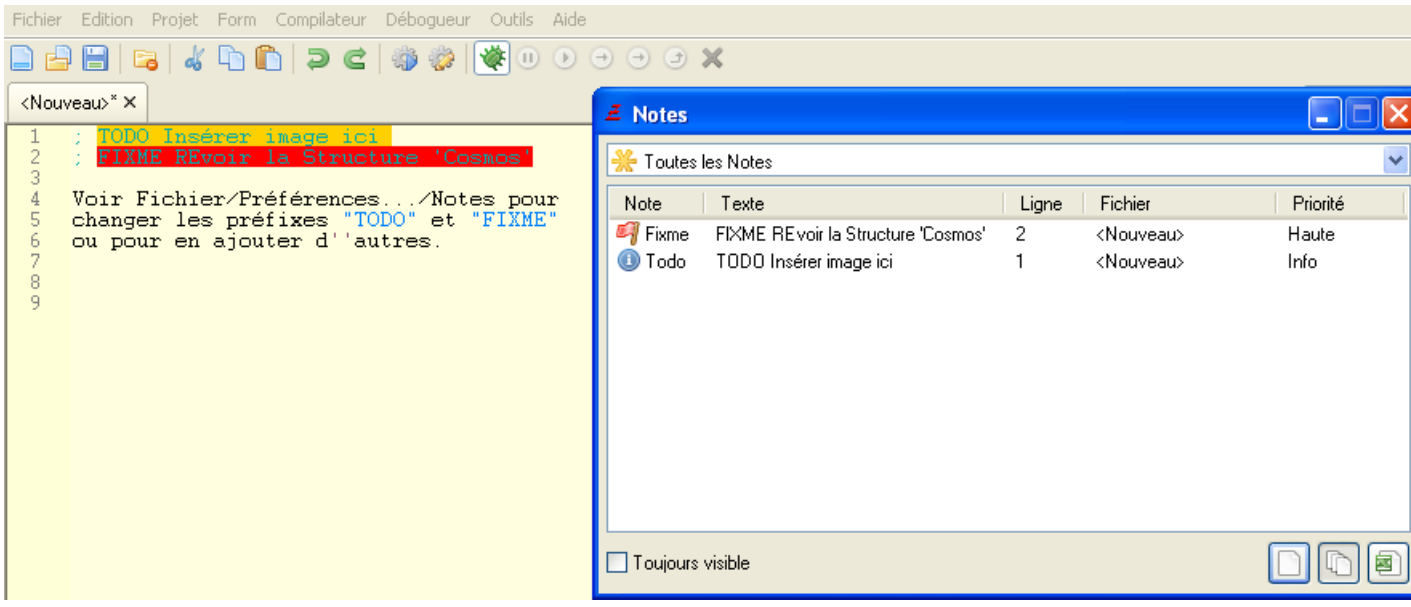


Le visualisateur de variables peut en fait afficher les variables, tableaux, listes, constantes, structures et interfaces définis dans le source en cours d'édition, ou dans tous les fichiers ouverts. La configuration de ce qui doit être affiché se fait dans les préférences .
Note : l'affichage des variables est un peu limité pour l'instant. Seules les variables qui sont déclarées par Define , Global , Shared , Protected ou Static seront reconnues.

Modèles de codes

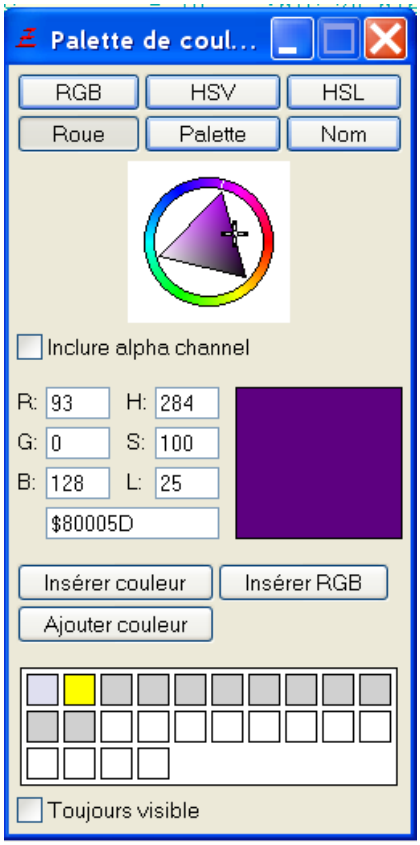
Cet outil permet d'organiser de manière hiérarchisée une liste de petits bouts de code qui sont souvent utilisés. Ils peuvent être insérés rapidement à n'importe quel endroit du fichier en cours d'édition en double-cliquant sur le code voulu.

Explorateur de Notes



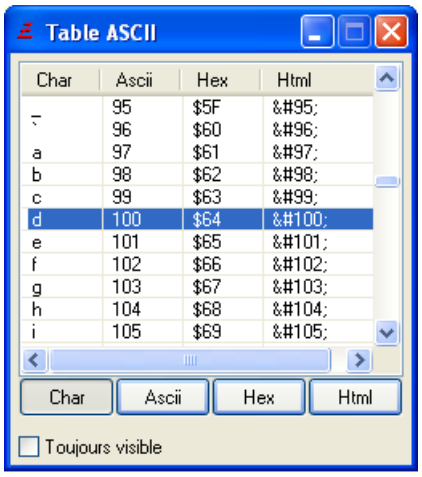
L'outil Explorateur de Notes recueille les commentaires du code source qui correspondent au format défini et les répertorie par ordre de priorité. Il peut être utilisé pour suivre les zones du code source qui doivent encore être travaillées.
Chaque problème affiché correspond à un commentaire dans le code. Un double-clic sur la Note affiche la ligne de code. Les Notes peuvent être affichées pour le fichier en cours, ou pour plusieurs fichiers (tous les fichiers ouverts, ou tous les fichiers qui appartiennent aux projet courant). La liste peut également être exportée au format CSV.
Pour configurer la liste des Notes recueillies, consultez la section "Notes" dans les Préférences .

Choix de couleur



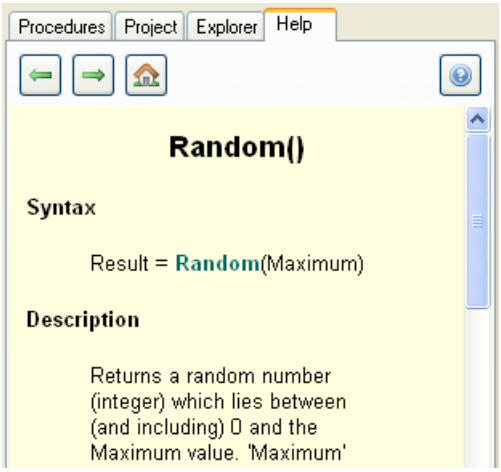
Le sélecteur de couleur vous aide à trouver la valeur de la couleur parfaite pour n'importe quelle tâche.
Les méthodes suivantes de choix de couleur sont :
RGB : Sélectionnez une couleur en choisissant les intensités de rouge, de vert et de bleu.
HSV : Sélectionnez une couleur en choisissant la teinte, la saturation et la valeur.
HSL : Sélectionnez une couleur en choisissant la teinte, la saturation et la luminosité.
Roue : Sélectionnez une couleur en utilisant le modèle HSV dans la roue de couleur.
Palette : Sélectionnez une couleur dans une palette prédéfinie.
Nom : Sélectionnez une couleur dans une palette par son nom.
La sélection des couleurs comprend une composante alpha, si l'option "Inclure le canal alpha" est activée. Les composantes individuelles (intensités rouge / vert / bleu ou teinte / saturation / luminosité), ainsi que la représentation hexadécimale de la couleur actuelle peuvent être vues et modifiées dans les champs de texte.
Le bouton "Insérer Couleur" insère la valeur hexadécimale de la couleur courante dans le code source. Le bouton "Insérer RGB" insère la couleur comme le ferait un appel à la fonction RGB() ou RGBA() dans le code. Le bouton "Ajouter couleur" permet d'ajouter la couleur actuelle dans la zone d'historique. En cliquant sur une couleur dans cette zone, permet à cette couleur de devenir la couleur courante.

Table des caractères



La table affiche une liste des 256 premiers caractères unicode avec leurs correspondances en décimal, hexadécimal et HTML. En double-cliquant sur une ligne, ce caractère sera inséré dans le code source. Les boutons en dessous de la liste permettent de choisir le format dans lequel le caractère sera inséré.

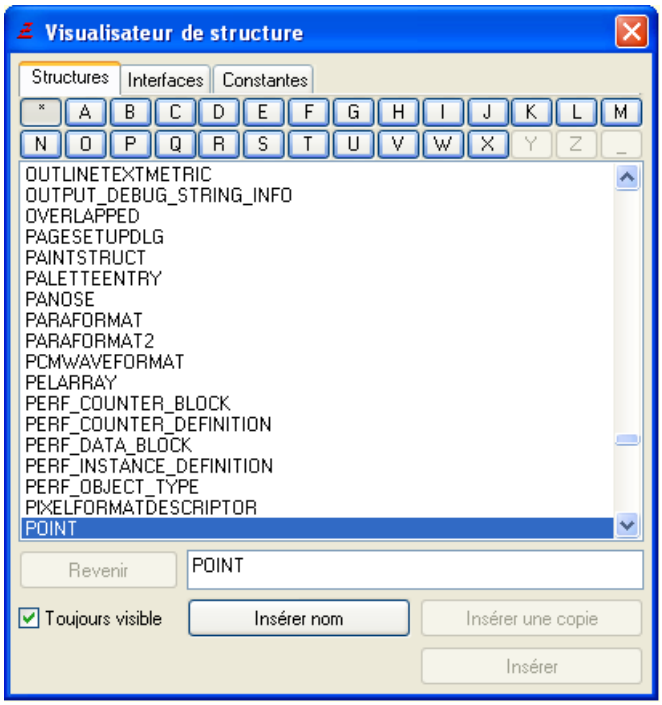
Outil Aide



L'outil d'aide est une visionneuse alternative pour le guide de référence . Il peut être utilisé pour afficher le manuel de PureBasic à côté du code, que ce soit avec le raccourci F1 ou non. Voir préférences .

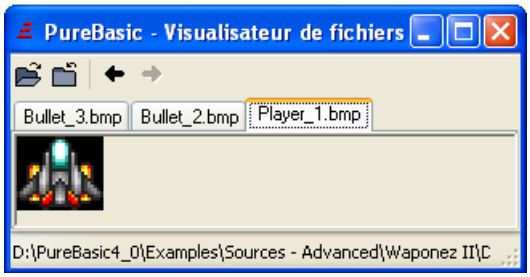
Les autres outils intégrés

Visualisateur de structures



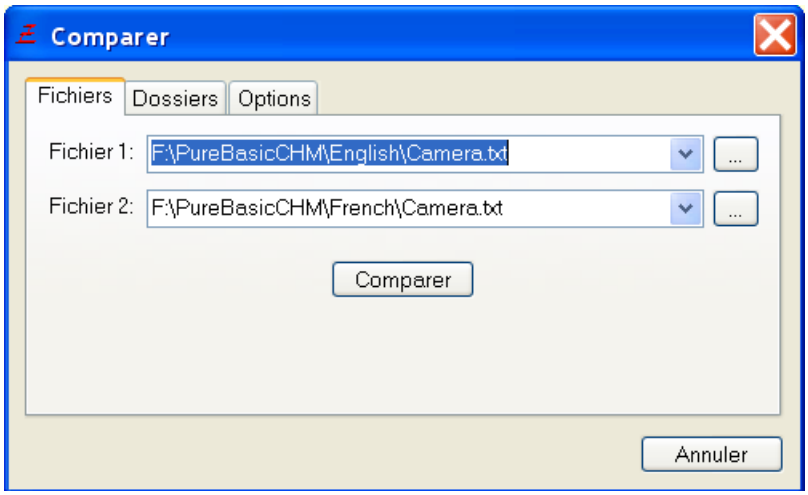
Cet outil permet de voir toutes les structures, interfaces et constantes qui sont prédéfinies dans PureBasic. Double-cliquer sur une structure ou une interface affichera la déclaration (le contenu) de l'élément. Il est possible de filtrer l'affichage en choisissant une lettre dans les boutons affichés au dessus de la liste.
Le bouton "Retour" revient à l'affichage précédant le double-clic.
Le bouton "Insérer nom" insère uniquement le nom de l'élément sélectionné.
Le bouton "Insérer copie" insère une copie de la déclaration de l'élément sélectionné.
Le bouton "Insérer" permet d'entrer le nom d'une variable et d'insérer tous les champs de la structure ou interface sélectionnée (en utilisant cette variable comme base).

Visualisateur de fichiers

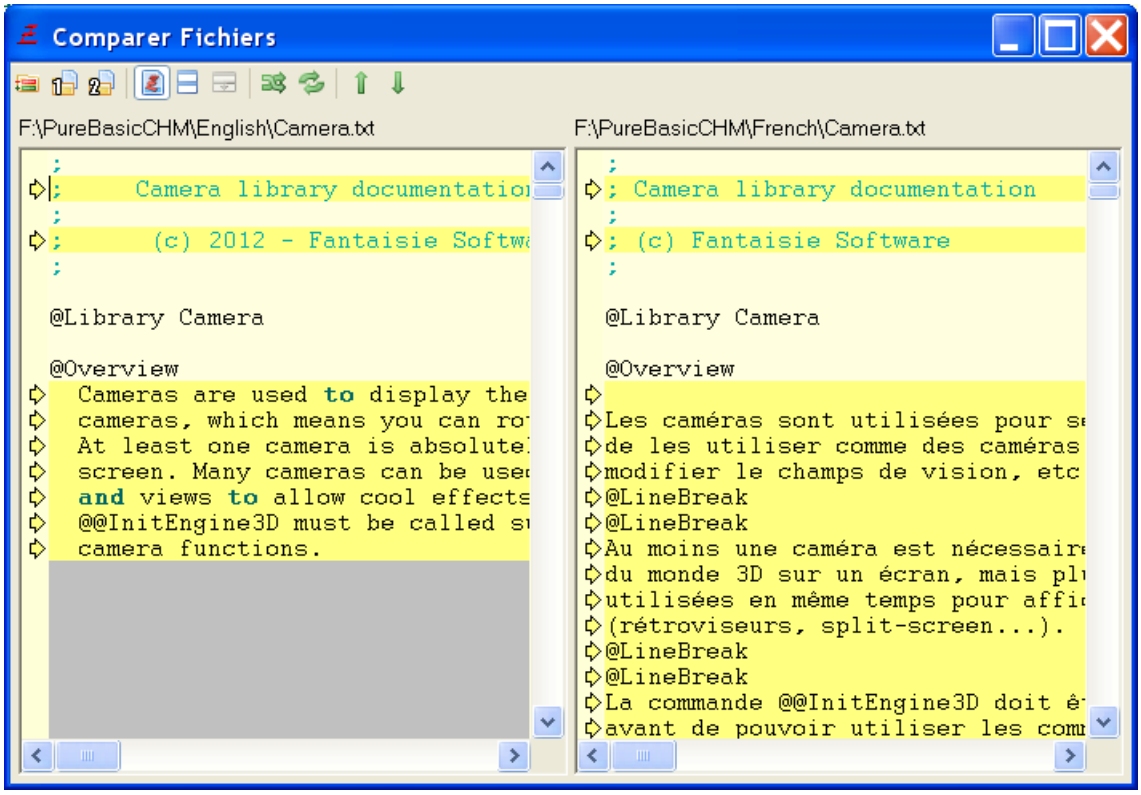


Le visualisateur de fichiers permet d'afficher plusieurs types de fichiers : textes, images et pages web (Windows seulement). Si un type de fichiers n'est pas reconnu, il sera affiché sous forme hexadécimale.
Le bouton "Ouvrir" ouvre un nouveau fichier et le bouton "X" ferme le fichier courant. Les flèches permettent de naviguer parmi les fichiers ouverts.
A noter qu'il est aussi possible d'ouvrir des fichiers dans le visualisateur interne en double-cliquant sur des fichiers binaires dans l'outil 'Explorer' ou sur le mot clef IncludeBinary dans la zone d'édition.

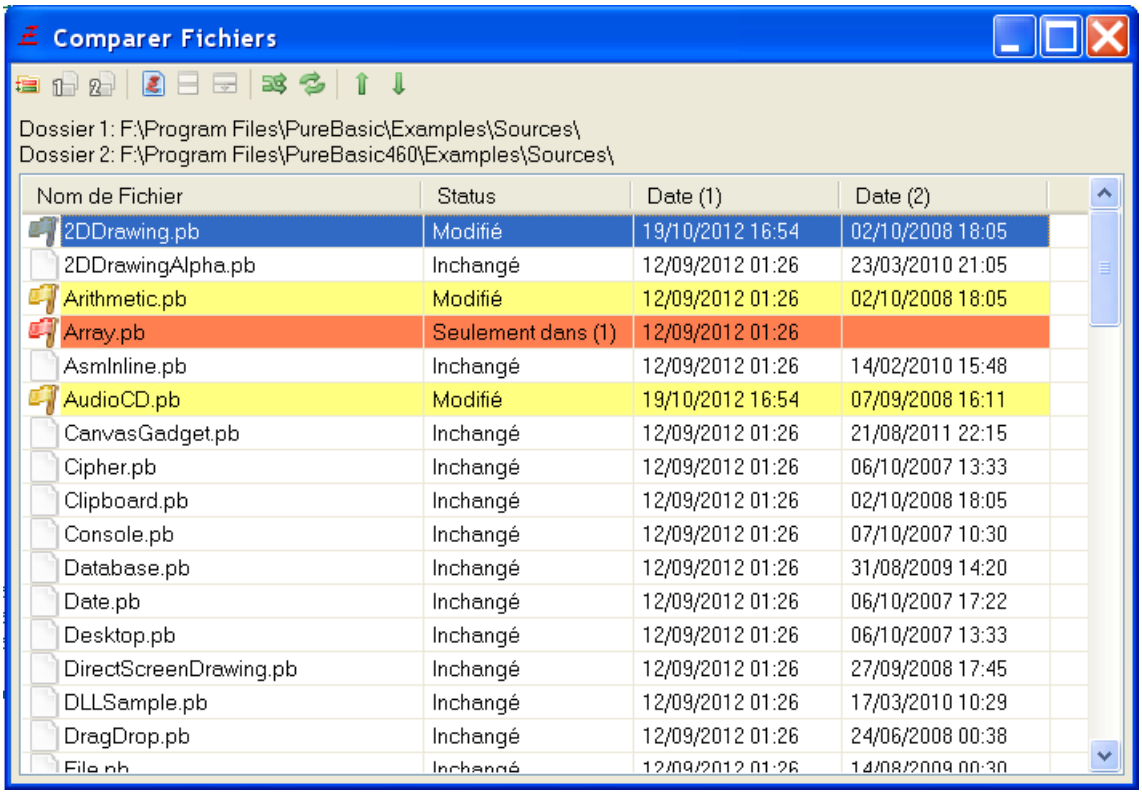
Comparer Fichiers/Dossiers



Cet outil permet de comparer deux fichiers (texte) ou des répertoires et de mettre en évidence leurs différences. L'onglet "Options" peut être utilisé pour ignorer des différences telles que les espaces ou les changements majuscules/minuscules.



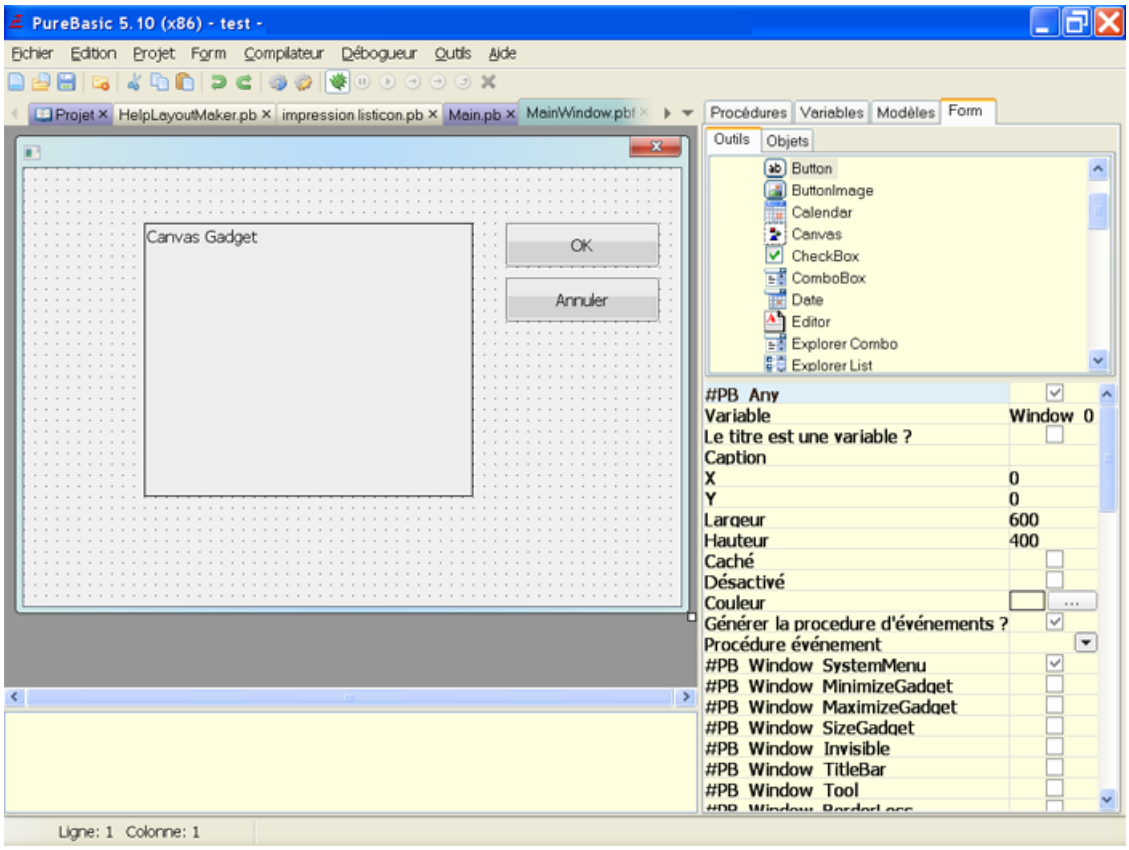
Les fichiers sont affichés côte à côte avec les différences marquées de la manière suivante : Les lignes affichées en rouge ont été enlevées dans le fichier, les lignes indiquées en vert ont été ajoutées dans le fichier et les lignes indiquées en jaune ont été modifiées.



Dans la comparaison de répertoires, le contenu des deux répertoires est examiné (avec la possibilité de filtrer la recherche par extension de fichier et d'inclure les sous-répertoires) et les fichiers sont marqués d'une manière similaire : Dossiers en rouge n'existent pas dans le deuxième répertoire, les fichiers en vert sont nouveaux dans le deuxième répertoire et fichiers en jaune ont été modifiés. Un double-clic sur un fichier modifié montre les modifications qui ont été apportées.

Autes outils du menu Outils

Concepteur de fenêtre

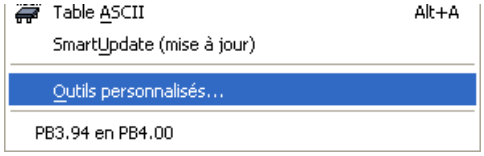


Le concepteur de fenêtre (ou de formulaire) peut être utilisé pour concevoir l'interface utilisateur de votre application. Pour plus d'informations, reportez-vous au chapitre concepteur de fenêtre .

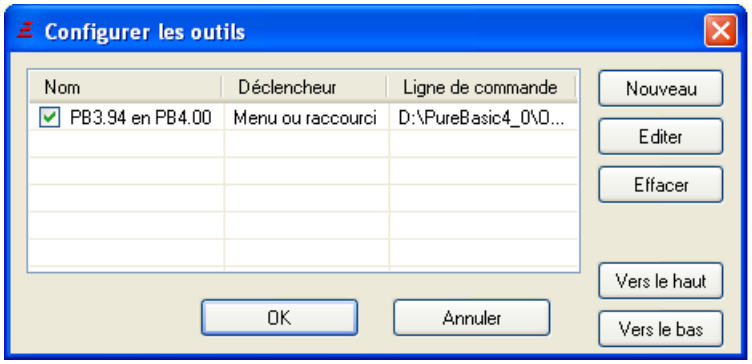
Chapitre 16

Les outils externes

L'IDE du PureBasic vous permet de configurer des programmes externes pour qu'ils soient appelés directement à partir d'un menu, de raccourcis, de la barre d'outils ou d'évènements "déclencheurs" spéciaux (ex : ouverture de fichier, fermeture etc..). Ceci permet un accès aisé à ces outils tout en programmant.
Vous pouvez également écrire vos propres petits outils en PureBasic pour effectuer des actions spéciales sur le code source que vous êtes en train d'éditer, afin d'automatiser les tâches récurrentes (ex : réorganisation du code, statistiques, etc..). De plus, vous pouvez configurer des visualisateurs de fichiers externes pour remplacer le visualisateur de fichier intégré à l'IDE pour des types particuliers de fichier ou pour tous les fichiers.



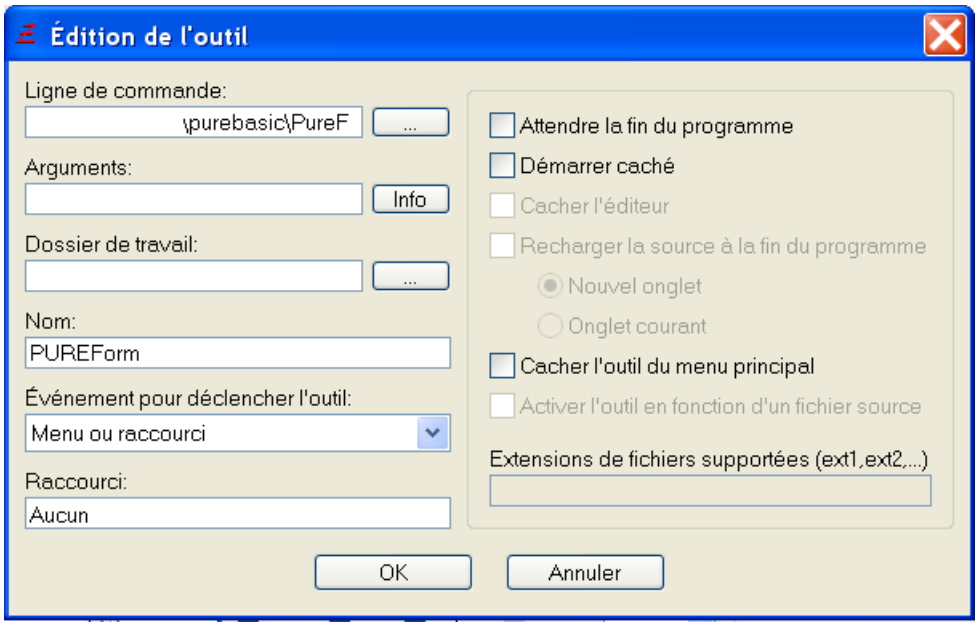
La commande "Outils personnalisés" du menu "Outils" ouvre la fenêtre de gestion des outils externes. La liste affiche tous les outils déjà configurés par ordre d'apparition dans le menu "Outils" (les outils ayant l'option 'caché' figurent aussi). Vous pouvez ajouter (bouton Nouveau) et supprimer (bouton Effacer) des outils, ou modifier leur ordre de priorité en cliquant sur les boutons "Vers le haut" / "Vers le bas" après avoir sélectionné un élément.



Chaque outil peut être rapidement activé ou désactivé à partir de la fenêtre "Outils personnalisés" du menu "Outils", en cochant ou décochant la case à gauche du nom de l'outil.

Configurer un outil

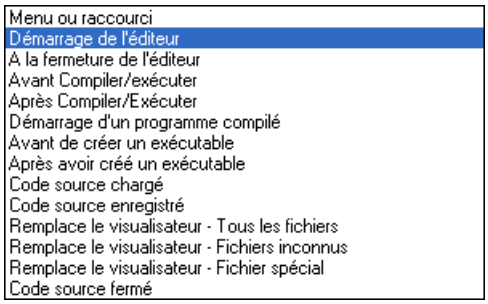
Les seuls éléments obligatoires dans la configuration sont la ligne de commande du programme à exécuter et son nom dans la liste du menu "Outils". Tout le reste est optionnel.



Ligne de commande
Contient le chemin et le nom du programme à exécuter.
Arguments
Contient les arguments qui seront passés au programme. Vous pouvez écrire des options fixes, tout comme des mots-clefs spéciaux qui seront remplacés lors de l'exécution du programme) :
%PATH : sera remplacé par le chemin du fichier source en cours d'édition. Il sera vide si le code source n'a pas encore été enregistré.
%FILE : sera remplacé par le nom du fichier source en cours d'édition. Il sera vide si le code source n'a pas encore été enregistré. Si l'outil est destiné à remplacer le visualisateur de fichier intégré, ce mot-clef représentera le fichier à ouvrir.
%TEMPFILE : quand cette option est spécifiée, le fichier source en cours d'édition est enregistré dans un fichier temporaire dont le nom est inséré ici. Ce fichier est créé uniquement pour l'outil et peut être librement modifié ou effacé.
%COMPILEFILE : ce mot-clef est uniquement valide pour les déclencheurs de compilation (voir ci-dessous). Il sera remplacé par le nom du fichier temporaire qui est envoyé au compilateur pour effectuer la compilation. En modifiant ce fichier, il est possible de changer ce qui sera effectivement compilé.
%EXECUTABLE : sera remplacé par le nom de l'exécutable créé par la dernière commande "Créer un exécutable". Pour le déclencheur "Après Compiler/Exécuter", il sera remplacé par le nom du fichier exécutable temporaire créé par le compilateur.
%CURSOR : sera remplacé par la position actuelle du curseur dans le code en cours d'édition, sous la forme : LIGNExCOLONNE.
%SELECTION : sera remplacé par la sélection actuelle sous la forme : LIGNEDEBUTxCOLONNEDEBUTxLIGNEFINxCOLONNEFIN. Il peut être utilisé en conjonction avec %TEMPFILE, si vous voulez que votre outil effectue des actions basées sur le texte sélectionné.
%WORD : sera remplacé par le mot actuellement situé sous le curseur.
%PROJECT : le chemin complet vers le répertoire contenant le fichier du projet si un projet est ouvert.
%HOME : le chemin complet vers le répertoire PureBasic. Note : pour chaque mot-clef désignant un fichier ou un chemin, il est généralement conseillé de les placer entre "" (ex : "%TEMPFILE") pour s'assurer que les chemins contenant des espaces seront correctement transmis à l'outil. Ces mots-clefs ainsi qu'une brève description peuvent aussi être consultés en cliquant sur le bouton "Info" à droite du champ Arguments.
Dossier de travail
Permet de choisir un répertoire que l'outil utilisera lors de son lancement. Si aucun répertoire n'est indiqué, l'outil utilisera le répertoire du code source en cours d'édition.
Nom
Permet d'attribuer un nom à l'outil. Ce nom s'affichera dans la liste des outils, et si l' outil n'est pas caché, dans le menu "Outils".

Evènement pour déclencher l'outil

Ici vous pouvez choisir quand l'outil devra être lancé. Plusieurs outils peuvent avoir le même déclencheur, ils seront tous exécutés quand cet évènement déclencheur se produira. L'ordre de leur exécution dépend de l'ordre dans lequel ils apparaissent dans la liste des outils.



Menu ou raccourci
L'outil ne sera pas lancé automatiquement. Il sera exécuté à partir d'un raccourci clavier ou d'un menu. Note : pour exécuter un programme à partir de la barre d'outils, il faut lui ajouter un bouton dans Fichier/Préférences/Général/Barre d'outils (voir Configurer l'IDE pour plus d'informations).
Si ce déclencheur est choisi, l'option "Raccourci" devient active et permet d'entrer un raccourci qui lancera cet outil.
Démarrage de l'éditeur
L'outil sera exécuté immédiatement après le démarrage complet de l'IDE.
A la fermeture de l'éditeur
L'outil sera exécuté juste avant que l'IDE ne quitte. Notez que tous les fichiers sources auront déjà été fermés.
Avant Compiler/Exécuter
L'outil sera exécuté juste avant la compilation du fichier source. En utilisant le mot-clef %COMPILEFILE, il est possible de modifier le code à compiler. Ceci permet par exemple de créer un petit pré-processeur pour le code source. Notez que vous devriez activer l'option "Attendre la fin du programme" si vous voulez que les modifications soient prises en compte par le compilateur.
Après Compiler/Exécuter
L'outil sera exécuté juste après la compilation, mais avant que le programme ne soit executé. En utilisant le mot-clef %EXECUTABLE, il est possible de récupérer le nom du fichier qui vient d'être généré. Les modifications sont autorisées, mais si le fichier est effacé, alors une erreur surviendra lorsque l'IDE essaiera d'exécuter le fichier.
Démarrage d'un programme compilé
L'outil sera lancé lorsque la commande "Exécuter" du menu "Compilateur" est activée. L'outil est lancé avant que le programme ne soit exécuté. Le mot-clef %EXECUTABLE est valide ici aussi.
Avant de créer un exécutable
Cet évènement est identique à "Avant Compiler/Exécuter", mais il est déclenché juste avant la création de l'exécutable final.
Après avoir créé un exécutable
L'outil sera lancé une fois que l'exécutable final aura été créé. Le mot-clef %EXECUTABLE peut servir à récupérer le nom de l'exécutable créé et ainsi effectuer des actions dessus (ex : pour le compresser).
Code source chargé
L'outil se lancera à chaque fois qu'un code source sera chargé dans l'IDE. Les mots-clefs %FILE et %PATH sont toujours valides, car le fichier est forcément chargé à partir d'un média (disque, réseau etc..).
Code source enregistré
L'outil se lancera à chaque fois qu'un code source sera enregistré par l'IDE. Les mots-clefs %FILE et %PATH sont toujours valides, car le fichier vient d'être enregistré sur un média (disque, réseau etc.).
Code source fermé
L'outil se lancera à chaque fois qu'un code source sera sur le point d'être fermé. A ce stade le fichier est toujours là, donc vous pouvez obtenir son contenu avec le mot-clef %TEMPFILE. Le mot-clef %FILE sera vide si le fichier n'a jamais été enregistré.
Remplace le visualisateur - Tous les fichiers
L'outil remplacera complètement le visualisateur de fichier intégré. Si on essaie d'ouvrir un fichier qui ne

peut être édité dans l'IDE, l'IDE va d'abord essayer les outils externes ayant comme déclencheur ce type de fichier particulier, et si aucun ne correspond, alors le fichier sera géré par cet outil. Utilisez le mot-clef %FILE pour récupérer le nom du fichier qui doit être ouvert.
Note : un seul outil peut être associé à ce déclencheur. Tous les autres outils associés à ce déclencheur seront ignorés.
Remplace le visualisateur - Fichiers inconnus
Cet outil remplacera le visualisateur hexadécimal intégré, qui est normalement utilisé pour ouvrir les fichiers de types inconnus. Il sera exécuté, lorsque l'extension du fichier est inconnue pour l'IDE, et si aucun autre outil externe n'a été configuré pour gérer ce fichier (si un outil est configuré avec le déclencheur "Remplace le visualisateur - Tous les fichiers", alors cet outil ne sera jamais appelé).
Note : un seul outil peut être associé à ce déclencheur.
Remplace le visualisateur Fichier spécial
Ce déclencheur permet à l'outil de gérer des extensions de fichiers spécifiques. Il a une plus haute priorité que les évènements "Remplace le visualisateur - Tous les fichiers", "Remplace le visualisateur Fichiers inconnus" et plus haute également que le visualisateur de fichier intégré. Indiquez les extensions que l'outil doit gérer dans le champ prévu à cet effet, sur la droite. Plusieurs extensions peuvent être attribuées.
Une utilisation courante de ce déclencheur est, par exemple, la configuration d'un programme comme Acrobat Reader pour gérer les fichiers ayant l'extension "pdf". Ce qui permet d'ouvrir facilement ces fichiers à partir de l'Explorateur, du visualisateur de fichier intégré ou en double-cliquant sur le mot-clef "IncludeBinary" dans le source.

Autres options sur le côté droit

Attendre la fin du programme
L'IDE sera complètement bloqué jusqu'à ce que l'outil termine son exécution. Cette option est requise si l'outil doit modifier un fichier source qui doit être rechargé par la suite, ou passé au compilateur par les déclencheurs de compilation.
Démarrer caché
L'outil sera lancé en mode invisible. Ne pas utiliser cette option si le programme nécessite une interaction avec l'utilisateur, car il ne pourra plus être fermé.
Cacher l'éditeur
Uniquement disponible si l'option "Attendre la fin du programme" est activée. Cache l'éditeur pendant l'exécution de l'outil.
Recharger la source à la fin du programme
Uniquement disponible si l'option "Attendre la fin du programme" est activée, et quand le mot-clef %FILE ou %TEMPFILE est utilisé dans le champ "Arguments".
Une fois que l'outil a terminé son exécution, l'IDE rechargera le fichier source dans l'éditeur. Il est possible de choisir si le fichier doit être rechargé dans l'onglet actuel ou dans un nouvel onglet.
Cacher l'outil du menu principal
N'affiche pas l'outil dans la liste du menu "Outils" de l'IDE. C'est utile pour les outils qui doivent uniquement être exécutés à partir de déclencheurs spéciaux, mais pas à partir du menu.
Activer l'outil en fonction d'un fichier source
Les outils avec cet option seront inscrits dans la liste "Exécuter des outils" dans les options du compilateur , et exécutés uniquement pour les sources où il est permis. Notez que lors de la désactivation de l'outil dans la fenêtre "outils de configuration", il sera désactivé au niveau global et ne pourra être lancé depuis une source, même s'il est activé ici.
Cette option n'est valable qu'avec :
- Avant Compiler/Exécuter - Après Compiler/Exécuter - Exécuter Programme compilé - Avant créer Exécutable - Après créer Exécutable - Code source chargé - Code source enregistré - Code source fermé
Extensions des fichiers supportées (ext1,ext2,...)
Uniquement disponible pour le déclencheur "Remplace le visualisateur - Fichier spécial". Permet de saisir la liste des extensions gérées.

Astuces pour l'écriture de vos propres outils de traitement

L'IDE fournit des informations supplémentaires pour les outils sous la forme de variables d'environnement. Elles peuvent être facilement lues par l'outil à l'aide des commandes de la bibliothèque Process .
Voici une liste des variables fournies. Notez que celles qui donnent des informations sur le fichier en cours d'édition ne sont pas accessibles pour les outils exécutés au démarrage ou à la fermeture de l'IDE.

```
PB_TOOL_IDE         - Chemin complet et nom de l'exécutable de l'IDE
PB_TOOL_Compiler    - Chemin complet et nom de l'exécutable du
 compilateur
PB_TOOL_Preferences - Chemin complet et nom du fichier des
 préférences de l'IDE
PB_TOOL_Project     - Chemin complet et le nom du projet actuellement
 ouvert (le cas échéant)
PB_TOOL_Language    - Langue actuellement utilisée dans l'IDE
PB_TOOL_FileList    - Liste de tous les fichiers ouverts dans l'IDE,
 séparés par un Chr(10)

PB_TOOL_Debugger    - Ces variables fournissent les paramètres
 correspondant à ceux de la fenêtre d'options de compilation

PB_TOOL_InlineASM     pour le source en cours. Elles sont à 1 si
 l'option
PB_TOOL_Unicode       est activée, à 0 sinon.
PB_TOOL_Thread
PB_TOOL_XPSkin
PB_TOOL_OnError

PB_TOOL_SubSystem   - Contenu du champ Bibliothèque Sous-système
dans les options de compilation
PB_TOOL_Executable  - Identique au mot-clef %COMPILEFILE pour la
 ligne de commande
PB_TOOL_Cursor      - Identique au mot-clef %CURSOR      pour la
 ligne de commande
PB_TOOL_Selection   - Identique au mot-clef %SELECTION   pour la
 ligne de commande
PB_TOOL_Word        - Identique au mot-clef %WORD        pour la
 ligne de commande

PB_TOOL_MainWindow  - Identifiant système (Handle) de la fenêtre
 principale de l'IDE
PB_TOOL_Scintilla   - Identifiant système (Handle) du composant
 d'édition de code Scintilla pour le source en cours
```

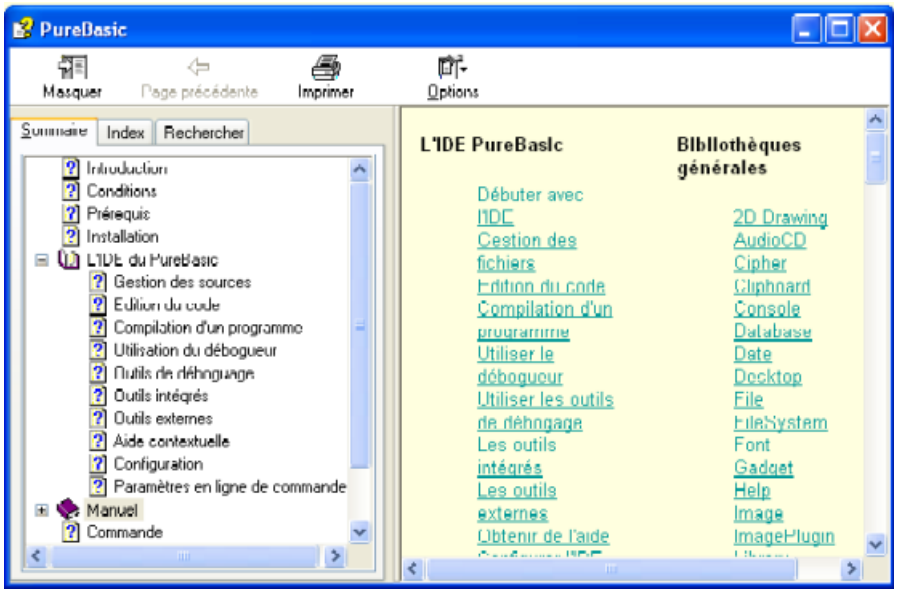
Quand les mots-clefs %TEMPFILE ou %COMPILEFILE sont utilisés, l'IDE ajoute les options de compilation à la fin du fichier temporaire (sous forme de commentaires), même si l'utilisateur a choisi de ne pas sauver ces options en enregistrant un code source.
Ceci permet à l'outil de lire les options de compilation spécifiques à ce fichier, et de les prendre en compte dans les actions qu'il va effectuer.

Chapitre 17

Obtenir de l'aide

L'IDE du Purebasic permet l'accès immédiat à l'aide de PureBasic, ainsi qu'aux autres fichiers d'aides disponibles sur la plateforme de développement sans quitter l'éditeur.

Accès rapide à l'aide de référence



En sélectionnant la commande "Aide..." du menu "Aide" ou en utilisant son raccourci clavier (par défaut F1) quand le curseur est sur un mot-clef ou une fonction PureBasic, l'aide sera ouverte à la page correspondante.
Si le mot à la position du curseur n'a pas de rubrique dédiée dans l'aide, la page de référence sera affichée.

Accès rapide à l'aide sur l'API Windows

Il y a deux méthodes pour accéder rapidement à l'aide des fonctions de l'API Windows supportées par PureBasic :
Le Microsoft Plateform SDK Ce SDK regroupe toute l'aide actuellement disponible pour le développement d'un programme sous Windows. Il contient des informations sur toutes les fonctions API, ainsi que des petits tutoriaux et des discussions techniques sur des points particuliers. De ce fait, il est plutôt conséquent (jusqu'à 400 Mo en fonction des composants installés).
Pour l'intégration avec l'IDE, il est possible d'installer l'édition de "Fevrier 2003" ou de "Windows Server 2003 SP1".

Le SDK peut être téléchargé à partir du lien suivant :
http ://www.microsoft.com/msdownload/platformsdk/sdkupdate/
A noter qu'il peut aussi être commandé par CD. Par ailleurs, il est aussi fourni dans la plupart des logiciels de développement de Microsoft (comme Visual Studio).
Le fichier d'aide Win32.hlp Il y a une autre alternative bien plus petite au SDK complet (7.5 Mo). Ce fichier d'aide est très ancien (écrit pour Windows 95), donc il ne contient pas d'informations concernant les API introduites depuis là. Néanmoins, il contient les informations essentielles à propos des API les plus utilisées, qui sont toujours d'actualité car elles n'ont quasiment pas évolué. Ce fichier est recommandé seulement pour une utilisation occasionnelle des API (et que le SDK n'est pas disponible).
Le fichier peut être téléchargé ici :
http ://www.purebasic.com/download/WindowsHelp.zip
Pour l'utiliser à partir de l'IDE, il suffit de créer le sous-répertoire "Help" dans le dossier "PureBasic" et d'y copier le fichier "Win32.hlp".

Accéder à d'autres fichiers d'aide

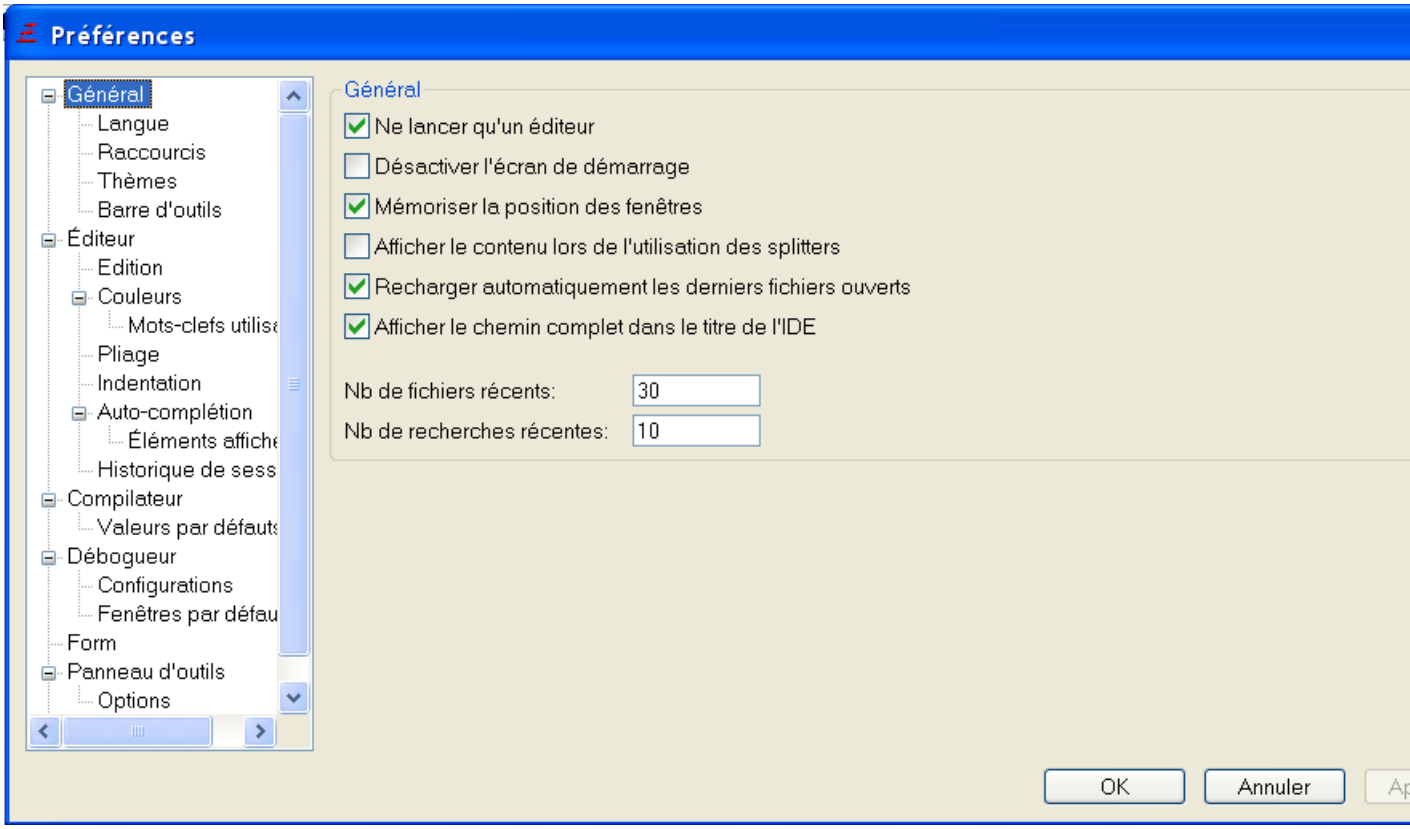
Pour accéder à d'autres fichiers d'aide à partir de l'IDE, il faudra créer le sous-répertoire "Help" dans le dossier "PureBasic" puis les copier dedans. Ces fichiers apparaîtront alors dans le sous-menu "Aide externe" du menu "Aide" (et dans le menu contextuel de la zone d'édition). Les fichiers .chm et .hlp seront affichés par les visualisateurs Microsoft. L'IDE ouvrira les fichiers d'aide dans le visualisateur interne. Les fichiers tels que les textes seront donc directement consultables. Pour les autres types, il faudra utiliser le menu Configuration des outils pour paramétrer un outil externe qui gérera ce type de fichier. L'aide sera alors affichée en utilisant cet outil.
Par exemple, si des fichiers d'aide sont en pdf, il suffit de paramétrer un outil externe qui ouvrira les pdf et mettre les fichiers pdf dans le sous-répertoire "Help" du dossier "PureBasic". Lorsqu'un de ces fichiers d'aide sera sélectionné dans le sous-menu "Aides externes", il sera ouvert en utilisant cet outil.

Chapitre 18

Configurer l'IDE

L'IDE du PureBasic propose un nombre très conséquent d'options pour qu'il puisse s'adapter aux habitudes des programmeurs qui l'utiliseront. Ces paramètres sont regroupés dans la fenêtre de la commande Préférences du menu "Fichier", et la description de chacune est décrite dans ce document. Tout changement ne sera effectif uniquement lorsque le bouton "OK" ou "Appliquer" sera utilisé.

Général



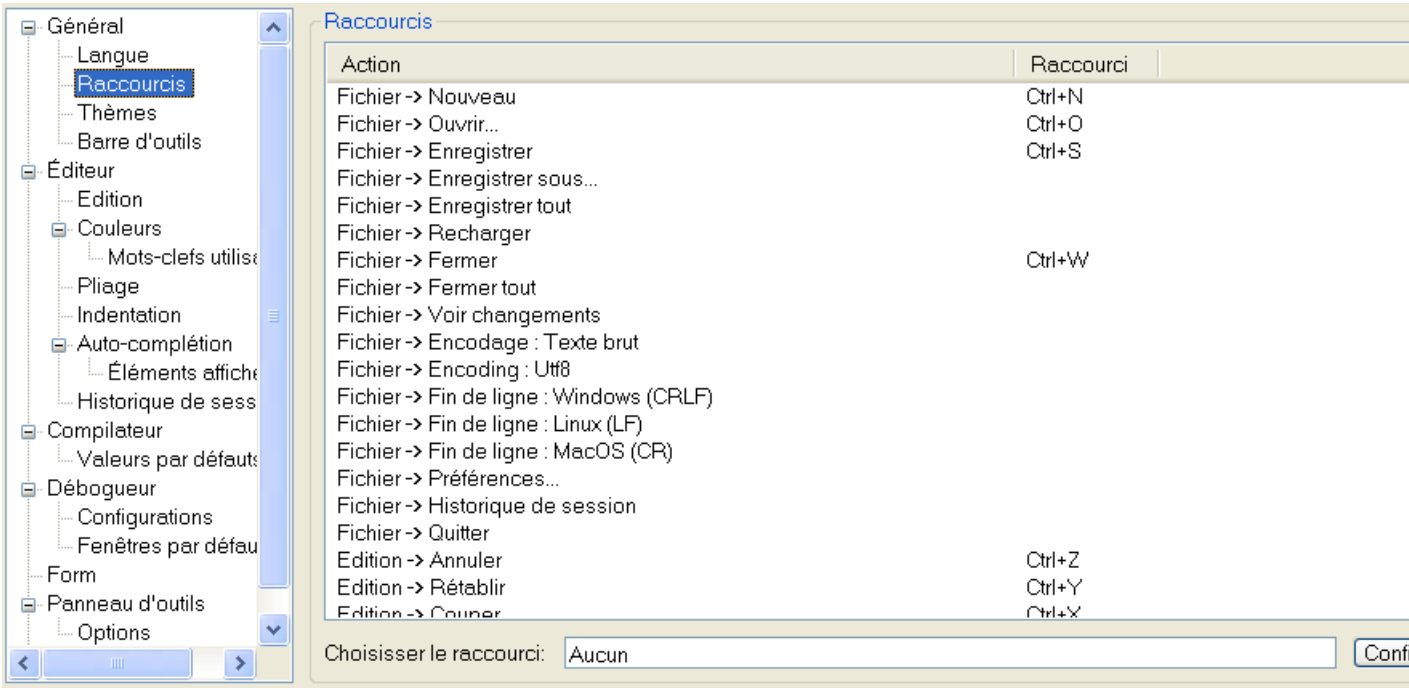
Cette section regroupe les options qui influent sur le comportement général de l'IDE.
Ne lancer qu'un éditeur
Si cette option est activée, l'IDE ne pourra être exécuté qu'une seule fois, et toute exécution ultérieure activera l'IDE déjà ouvert. Par exemple cliquer sur un fichier .pb dans l'explorateur ouvrira le fichier dans un nouvel onglet au lieu de lancer une nouvelle instance de l'IDE.
Désactiver l'écran de démarrage
Désactive l'écran "PureBasic" qui est affiché lors du démarrage de l'IDE.

Mémoriser la position des fenêtres
Enregistre la position et la taille des fenêtres de l'IDE lorsqu'elles sont fermées. Pour avoir les fenêtres toujours ouvertes avec les mêmes dimensions, il faudra les placer à l'endroit désiré, activer cette option, puis quitter l'IDE (pour enregistrer les options). Au prochain démarrage, désactiver cette option et les fenêtres utiliseront la taille et la position précédemment enregistrées.
Afficher le contenu lors de l'utilisation des splitters
Si l'ordinateur de développement est rapide, cette option peut être activée, sinon déplacer les splitters peut entrainer un scintillement désagréable.
Recharger automatiquement les derniers fichiers ouverts
Recharge automatiquement tous les fichiers qui étaient ouverts lorsque l'IDE a quitté pour la dernière fois.
Afficher le chemin complet dans le titre de l'IDE
Si cette option est activée, le titre de la fenêtre principale de l'IDE affichera le chemin complet du fichier en cours d'édition, sinon seul le nom de fichier sera affiché.
Liste des fichiers récents
Nombre de fichiers récents à afficher dans le sous-menu "Fichiers récents" du menu "Fichier".
Recherches récentes
Détermine le nombre de mots récents qui seront mémorisés dans les commandes "Rechercher/Remplacer" et "Rechercher dans les fichiers".
Vérifier les mises à jour
Indique combien de fois l'IDE doit vérifier sur le serveur de purebasic.com la disponibilité des nouvelles mises à jour. Une vérification de mise à jour peut également être effectuée manuellement à tout moment à partir du menu "Aide".
Vérifier les versions finales (releases)
Indique quelles sont les types de version qui provoqueront une notification si elles sont disponibles.

Général - Langue

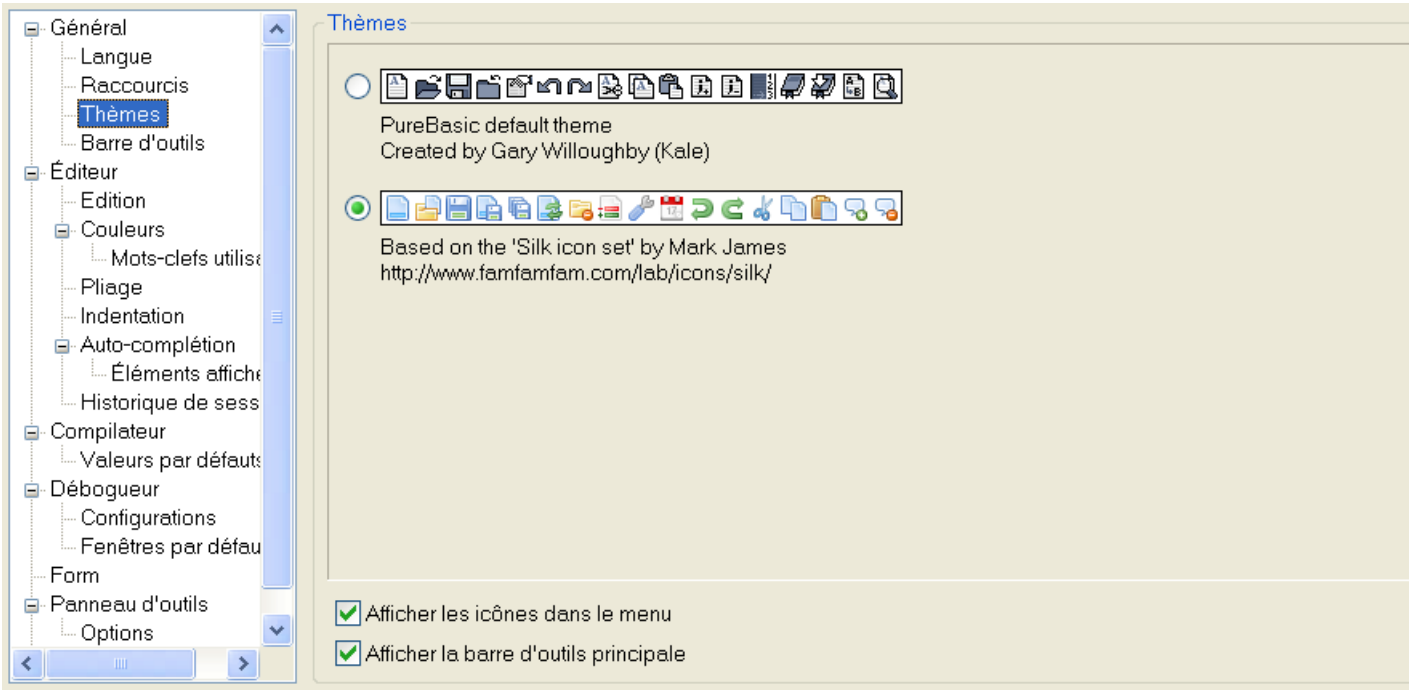
Permet de changer la langue utilisée par l'IDE. La liste déroulante affiche les langues disponibles et des informations relatives à la traduction (ex : la personne qui a fait la traduction et la date).

Général - Raccourcis claviers



Dans cette section il est possible de paramétrer absolument tous les raccourcis pour les commandes de l'IDE. Créer un nouveau raccourci, selectionner le champ de saisie du raccourci, taper directement le raccourci souhaité sur le clavier puis cliquer sur "Appliquer".
A noter que les touches Tab et Shift+Tab sont réservées pour l'indentation des blocs de code et ne peuvent pas être modifiées. De plus certaines combinaisons de touches peuvent avoir un sens particulier pour l'OS et ne pourront pas être utilisées.

Général - Thèmes



Cette section permet de visualiser les thèmes disponibles pour l'IDE et d'en sélectionner un. Par défaut, deux thèmes sont à disposition.
Davantage de thèmes peuvent être facilement ajoutés en créant un fichier zip contenant les images (au

format PNG) et un fichier "Theme.prefs" qui décrit le thème. Le fichier zip doit ensuite être copié dans le répertoire "Themes" de PureBasic pour être reconnu par l'IDE. Le fichier "SilkTheme.zip" peut être utilisé comme base pour créer un nouveau thème.

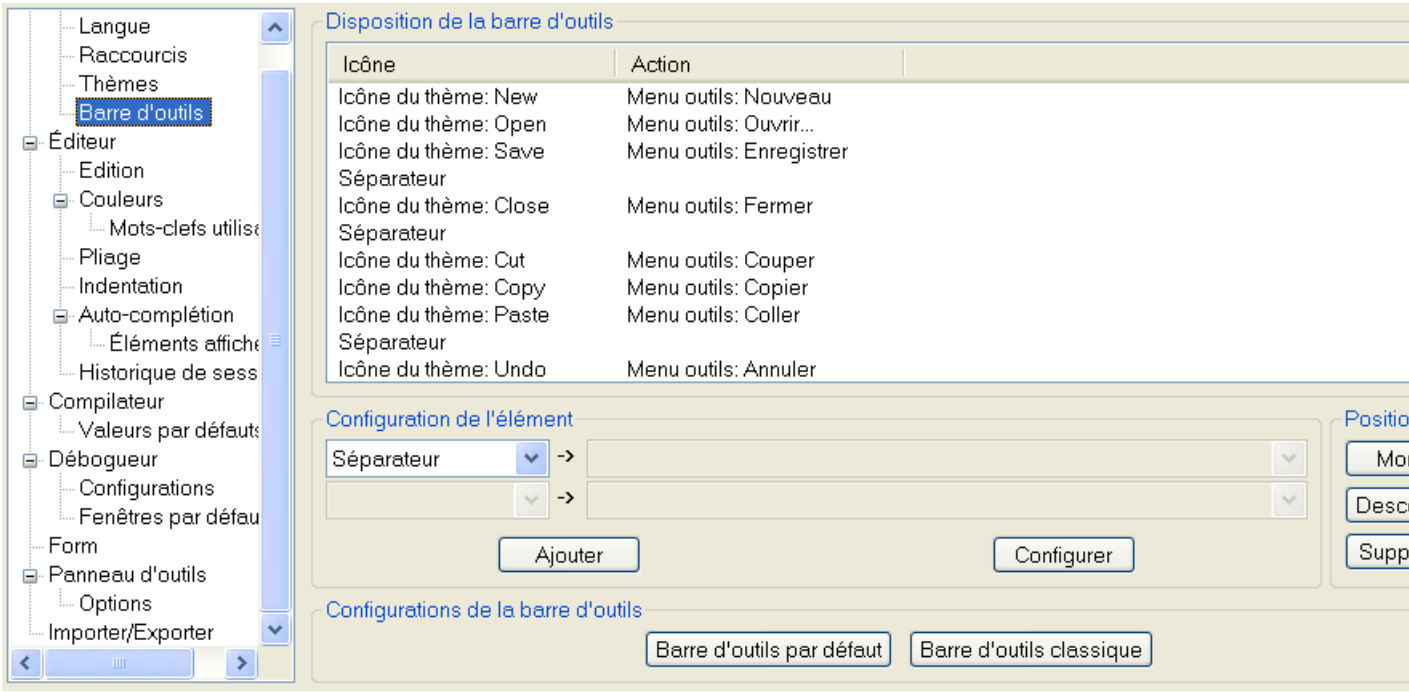
Afficher les icônes dans le menu

Permet d'afficher ou de cacher les icônes dans les menus de l'IDE.

Afficher la barre d'outils principale

Permet d'afficher ou de cacher la barre d'outils principale, pour gagner un peu de place dans la fenêtre d'édition.

Général - Barre d'outils



La barre d'outils est entièrement paramètrable. Pour changer la position des icônes, utiliser les boutons de la section "Position". Pour modifier un bouton ou en ajouter un nouveau, utiliser les boutons de la section "Configuration de l'élément" (les nouveaux éléments sont toujours ajoutés en fin de liste).

Types des éléments :

Séparateur : une barre verticale de séparation.

Espacement : un espace vide de la taille d'une icône.

Icône standard : permet de choisir une icône système (fournie par l'OS) dans la liste déroulante.

Icône de l'IDE : permet de choisir une icône propre à l'IDE dans la liste déroulante.

Fichier icône : permet de choisir une icône à partir d'un fichier spécifié dans le champ à droite (les fichiers PNG sont acceptés sur toutes les plateformes, ainsi que les fichiers icône sous Windows).

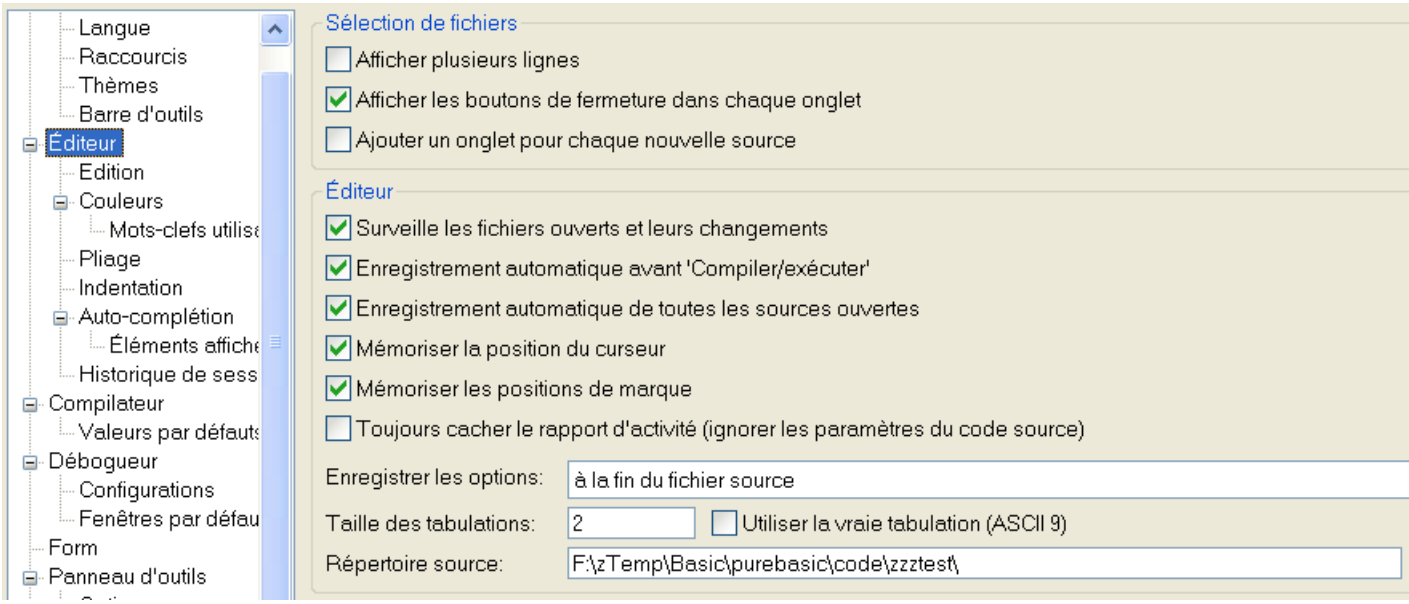
Si le type de l'élément n'est pas un "Séparateur" ni un "Espacement", alors il est possible de lui associer une action lorsque le bouton sera pressé :

Elément d'un menu : exécutera la commande correspondante à l'élément du menu spécifié dans la liste déroulante.

Lancer un outil : exécutera l'outil externe spécifié dans la liste déroulante.

La section "Configurations de la barre d'outils" contient les deux configurations de base de la barre d'outils qu'il est possible de choisir et de modifier à convenance.

Editeur



Paramètres qui influent sur l'édition des codes sources

Surveille les fichiers ouverts et leurs changements

Surveille toutes les modifications qui sont apportées aux fichiers sur le disque alors qu'ils sont édités dans l'IDE.

Si des modifications sont apportées par d'autres programmes, un avertissement s'affiche avec le choix de recharger le fichier à partir du disque.

Enregistrement automatique avant 'Compiler/Exécuter'

Enregistre le code source en cours d'édition avant chaque 'Compiler/Exécuter'. A noter que les fichiers inclus ne sont jamais enregistrés automatiquement.

Enregistrement automatique avant 'Créer un exécutable'

Enregistre le code source en cours d'édition avant de créer un fichier exécutable.

Enregistrement automatique de toutes les sources ouvertes

Enregistre toutes les sources et pas seulement la source en cours avec l'une des options de sauvegarde automatique .

Mémoriser la position du curseur

Enregistre la position du curseur ainsi que l'état de tous les replis pour le fichier en cours d'édition.

Mémoriser les positions des marques

Enregistre la position des marqueurs pour le fichier en cours d'édition.

Toujours cacher le rapport d'activité

Le journal d'erreurs est montré ou caché en fonction de la configuration de chaque code source. Cette option permet de s'assurer que le journal d'erreurs ne sera jamais affiché si l'utilisateur le souhaite. La partie du menu concernant le journal d'erreur est aussi supprimée.

Enregistrer les options

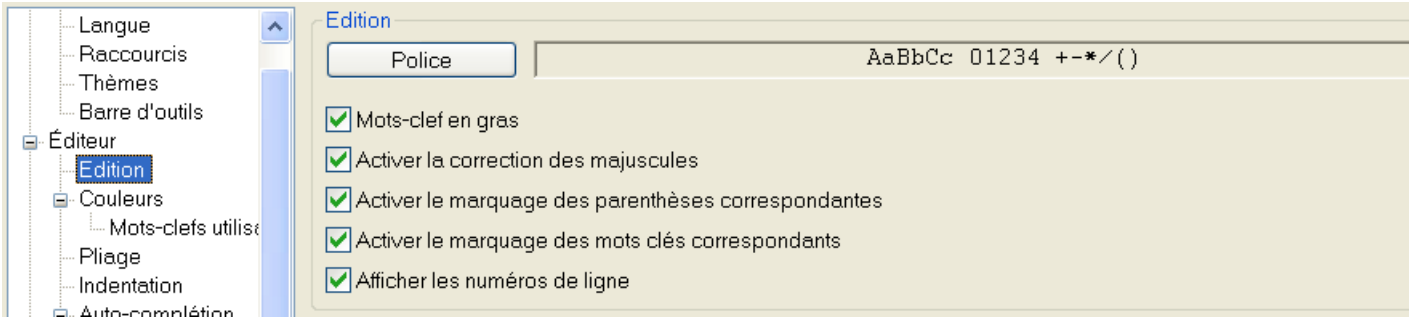
Permet de déterminer où les options propres à chaque fichier doivent être enregistrées :

> à la fin du fichier source
> Enregistre les options dans un bloc de commentaire à la fin du fichier. Lorsqu'un fichier contenant ce type de commentaires est ouvert par l'IDE, ils resteront invisibles.
> dans un fichier <nomdufichier>.pb.cfg
> Crée un fichier .pb.cfg pour chaque fichier enregistré qui contient ces informations.
> dans un fichier project.cfg pour chaque répertoire
> Crée un fichier nommé project.cfg dans chaque répertoire ou des fichiers PureBasic sont enregistrés. Ce fichier contiendra les options de tous les fichiers du répertoire.
> nulle part
> Les options ne sont pas enregistrées du tout. Quand les fichiers seront ouverts, ils utiliseront toujours les valeurs par défaut.

Valeur d'une tabulation

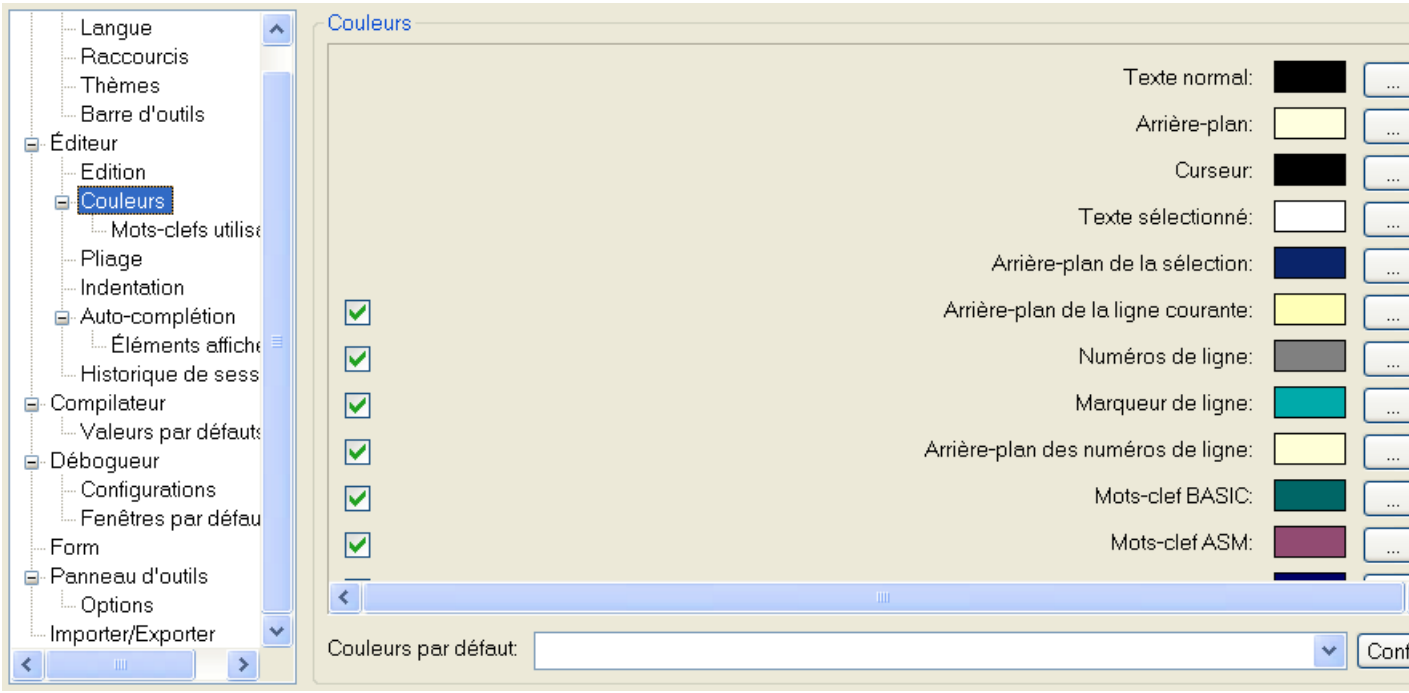
Permet de spécifier le nombre d'espaces qui seront insérés à chaque tabulation.
Utiliser les vraies tabulations (Ascii 9)
Si cette option est activée, les tabulations utiliseront le caractère 'tab' au lieu des espaces. A noter que si cette option est activée, le champ "Valeur d'une tabulation" définit la taille de la tabulation affichée.
Répertoire par défaut
Définit le répertoire par défaut qui sera utilisé par les commandes "Ouvrir" et "Enregistrer" si aucun fichier n'est ouvert, ou que le fichier en cours d'édition n'a pas encore été enregistré. Il est conseillé de mettre le répertoire où sont principalement stockés les codes sources.
Les extensions de fichier de code
L'IDE détecte les fichiers de code par leur extension (pb, pbi ou pbf par défaut). Les fichiers sans code sont modifiés dans un mode "texte brut" dans lequel les caractéristiques liées au code sont désactivées. Ce paramètre oblige l'IDE à reconnaître de nouvelles extensions de fichiers comme des fichiers de code. Le champ peut contenir une liste d'extensions séparées par des virgules (ex : "pbx, xyz").

Editeur - Edition



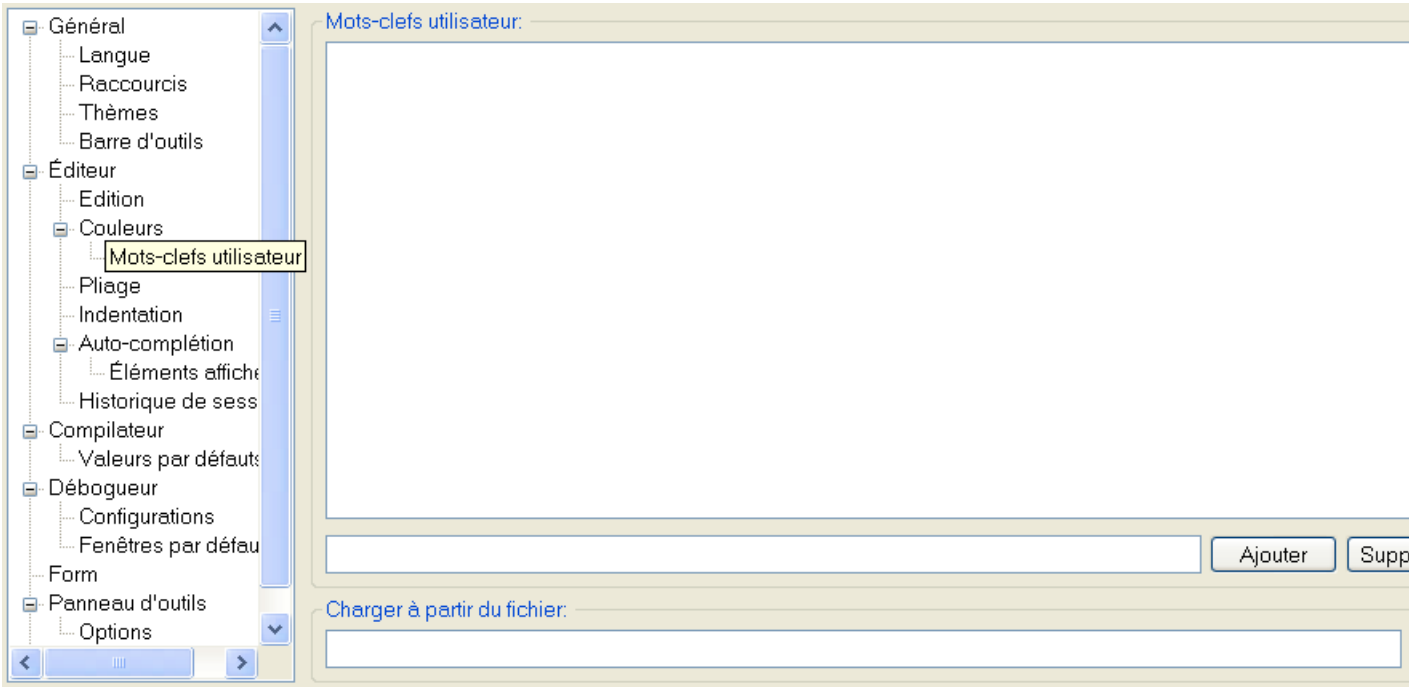
Utiliser "Sélectionner une police" pour changer la police de caractères utilisée pour l'affichage du code source. Pour assurer une bonne lisibilité du code, la police doit être de taille fixe (c'est à dire que tous les caractères ont la même largeur), si possible même pour les lettres en gras.
Activer la coloration syntaxique
Active ou désactive la coloration des codes sources. Le désactiver rend l'IDE légèrement plus rapide, mais bien moins convivial.
Mot clefs en gras
Si la police de caractères n'affiche pas les lettres en gras avec la même taille que les lettres normales, il est conseillé de désactiver cette option.
Activer la correction des majuscules
Corrige automatiquement les mots clefs PureBasic, les fonctions PureBasic et API ainsi que les constantes prédéfinies pour qu'ils soient exactement comme définis initialement (ex : 'openwindow()' sera automatiquement changé en 'OpenWindow()').
Activer le marquage des parenthèses correspondantes
Active le marquage des parenthèses lorsque le curseur se trouve sur une parenthèse. Cela permet de vérifier rapidement s'il y a le bon nombre de parenthèses, et de savoir à quelle parenthèse fermante correspond la parenthèse ouvrante (et vice-versa).
Activer le marquage des mots-clés correspondant
Permet de souligner tous les mots-clés en rapport avec le mot-clé actuellement sous le curseur.
Afficher les numéros de lignes
Affiche ou cache la numérotation des lignes située à gauche de la zone d'édition.

Editeur - Couleurs



Cette partie permet de modifier les couleurs utilisées par la coloration syntaxique ainsi que les marqueurs du débogueur. Des configurations prédéfinies sont disponibles dans la liste déroulante en bas de la fenêtre (une fois appliquées, elles sont modifiables). Chaque couleur peut être désactivée en utilisant les cases à cocher.
Note : Le modèle de couleur 'Accessibility' a (indépendamment des couleurs à fort contraste) un réglage spécial pour utiliser toujours la couleur système pour la sélection dans l'éditeur. Ceci permet aux applications qui permettent de lire un écran de mieux détecter le texte sélectionné.

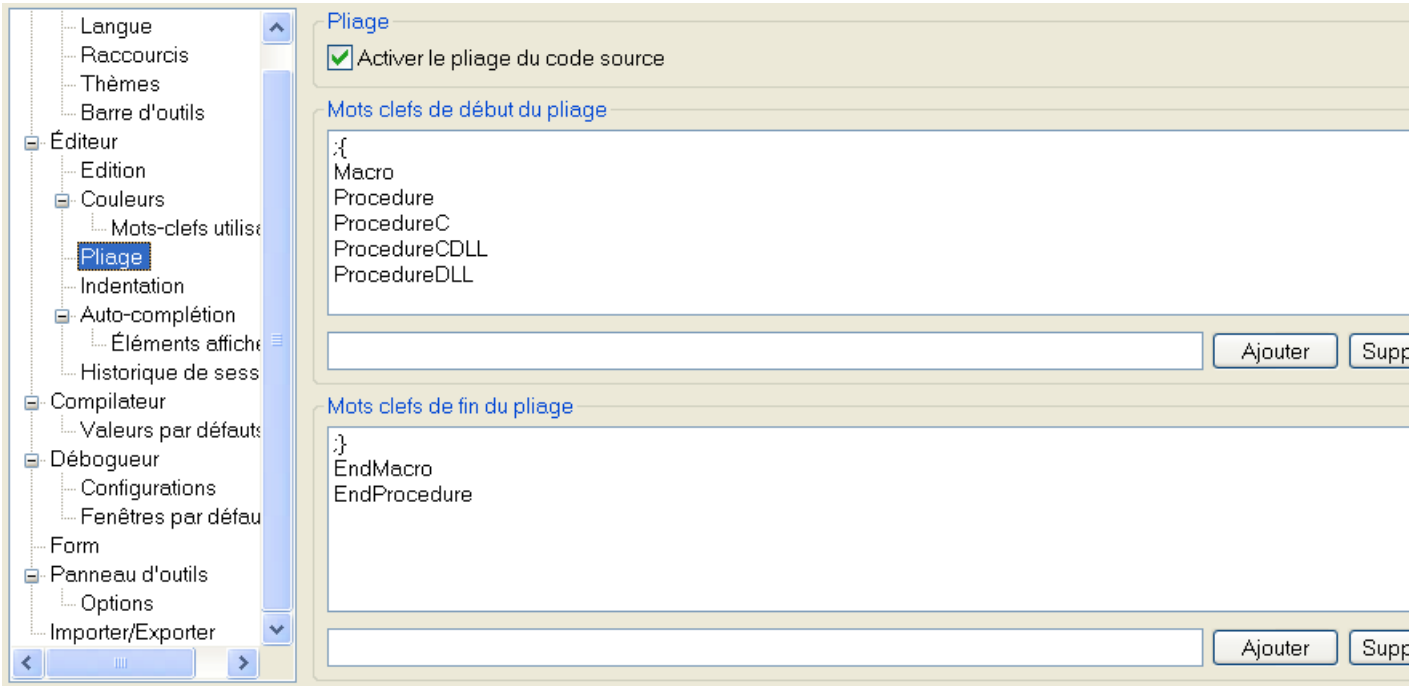
Editeur - Couleurs - Mots Clés personnalisés



Dans cette section, une liste de mots-clés personnalisés peut être définie. Ces mots-clés peuvent avoir une

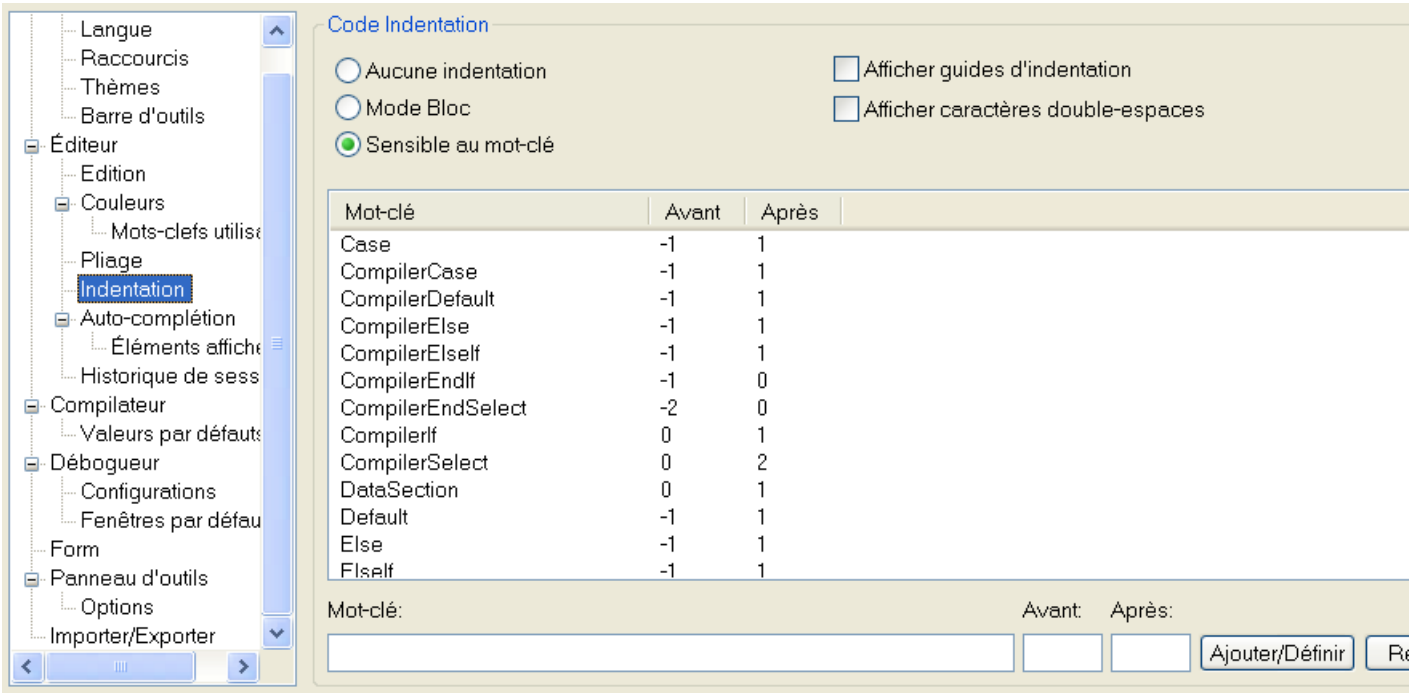
couleur spéciale qui leur est attribuée dans les options de couleur et l'IDE l'appliquera si cette fonction est activée. Cela permet d'appliquer une couleur particulière aux mots-clés par des outils-préprocesseurs ou des macros, ou tout simplement pour avoir des mots-clés PB colorés différemment.
Notez que ces mots-clés priment alors cela permet de changer la correction de couleur ou la casse même pour les mots-clés PureBasic.
Les mots-clés peuvent être entrés directement dans les préférences ou spécifiés dans un fichier texte avec un mot-clé par ligne. (ou les deux)

Editeur - Pliage



Il est possible d'ajouter des mots clefs pour marquer le début et la fin d'un bloc repliable dans le code source. Un nombre illimité de mots clefs peuvent être saisis. Les mots qui sont dans les commentaires sont ignorés sauf si le mot clef employé a pour initiale le symbole commentaire (comme le mot clef par défaut " ;{").
Un mot clef ne peut pas contenir d'espace.

Editeur - Indentation



Ici il est possible de paramétrer comment l'éditeur gère l'indentation du code quand la touche "Entrée" est appuyée.

Aucune indentation

Appuyer sur la touche "Entrée" place toujours le curseur au début de la ligne suivante.

Mode bloc

La nouvelle ligne aura la même indentation que la ligne courante.

Sensible au mot-clé

Appuyer sur la touche "Entrée" corrigera l'indentation de la ligne courante et de la nouvelle ligne, en fonction du mot-clé présent sur ces lignes. Les règles pour effectuer cette correction sont spécifiées dans la liste des mots-clés. Ces règles s'appliquent aussi lors d'un "Reformatter sélection" du menu édition .

Afficher les indentations

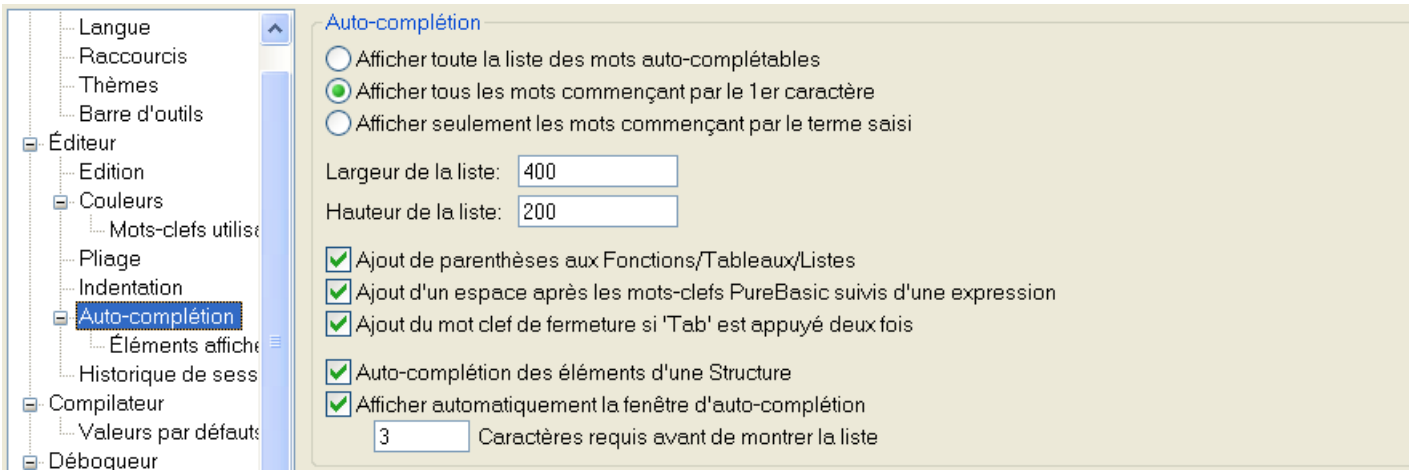
Affiche des lignes verticales pour visualiser les indentations de chaque ligne. Cela montre plus rapidement les lignes ayant le même niveau d'indentation.

Afficher les espaces

Affiche les caractères d'espacement sous forme de points (espaces) ou flèches (tabulations).

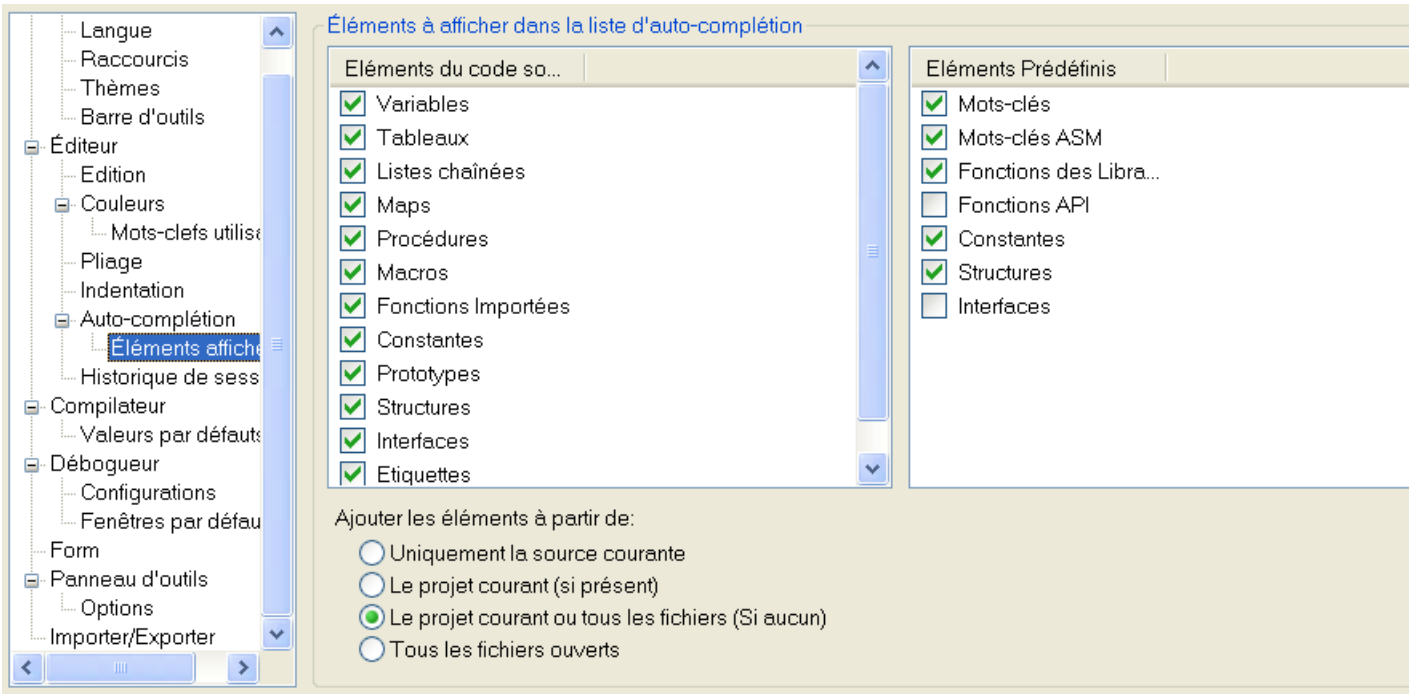
La liste des mots-clés contient les mot-clés qui influencent l'indentation. Le paramètre "Avant" indique le changement d'indentation sur le ligne qui contient le mot-clé, tandis que le paramètre "Après" spécfie le changement à appliquer sur la ligne suivante.

Editeur - Auto-complétion



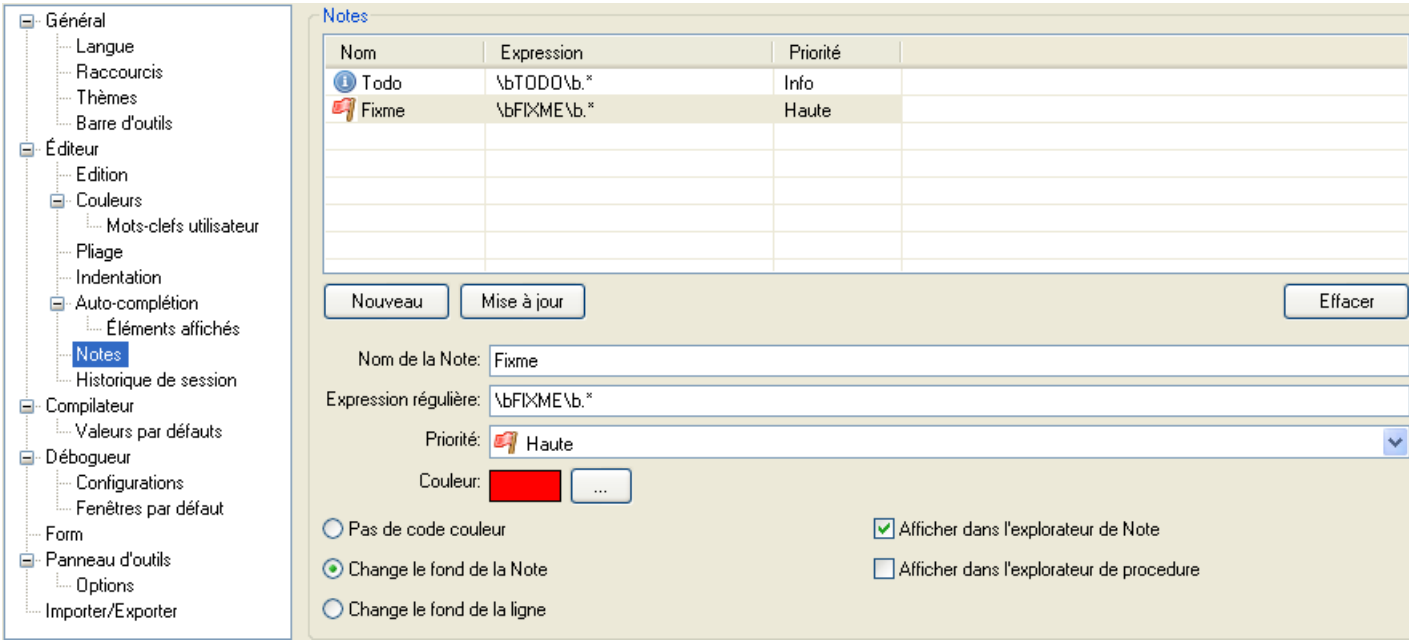
Afficher toute la liste des mots auto-complétables
Affiche toujours toute la liste des mots qui sont gérés par l'auto-complétion, mais choisit tout le même le mot le plus proche.
Afficher tous les mots commencant par le 1er caractère
Affiche uniquement les mots qui commencent par le premier caractère saisi. Le mot le plus proche est tout de même sélectionné.
Afficher seulement les mots commençant par le terme saisi
Affiche seulement les mots commençant par le terme saisi. Si aucun mot ne correspond, la liste ne sera pas affichée du tout.
Largeur de la liste / Hauteur de la liste
La dimension de la liste d'auto-complétion peut être définie ici (en pixels). Note : Ce sont les valeurs maximum. La fenêtre de la liste peut être plus petite s'il y a très peu d'éléments à afficher.
Ajout des parenthèses aux fonctions/tableaux/listes
Ajoute automatiquement la parenthèse ouvrante "(" après chaque fonction, tableau ou liste insérée par l'auto-complétion. Les fonctions sans paramètres et les listes chainées auront automatiquement les doubles parenthèses "()" ajoutées.
Ajout d'un espace après les mots-clefs PureBasic suivis d'une expression
Quand un mot clef PureBasic ne peut pas être utilisé tout seul (tel que If, For, While etc.) et nécessite une expression, un espace sera automatiquement ajouté lors de l'auto-complétion.
Ajout du mot clef de fermeture si Tabulation/Entrée est appuyée deux fois
Si la touche 'Entrée' ou 'Tabulation' est appuyée deux fois, le mot clef de fermeture sera automatiquement inséré (ex : "EndSelect" pour "Select", "EndIf " pour "If") après le mot clef lui-même (et après le curseur, dont il possible de continuer l'édition immédiatement après l'auto-complétion).
Auto-complétion des éléments d'une structure
Affiche automatiquement la liste d'auto-complétion quand le curseur est juste après une variable associée à une structure ou à une interface et que la touche "\" est entrée. La liste contiendra tous les champs qui composent la structure ou l'interface. Si cette option est désactivée, la liste peut toujours être affichée manuellement en utilisant le raccourci clavier associé (par défaut Ctrl+Espace).
Afficher automatiquement la fenêtre de complétion des fonctions
Affiche automatiquement la liste d'auto-complétion après qu'un nombre de caractères aient été entrés et qu'un ou plusieurs mots peuvent correspondre. Si cette option est désactivée, la liste peut toujours être affichée manuellement en utilisant le raccourci clavier associé (par défaut Ctrl+Espace).
Caractères requis avant l'affichage de la liste
Permet le configurer le nombre minimum de caractères saisis avant que la liste ne soit automatiquement affichée.

Editeur - Auto-complétion - Eléments affichés



Permet d'ajuster avec précision quels éléments seront affichés dans la liste d'auto-complétion.
Eléments du code source
Eléments définis dans le code source actuel, ou dans les autres sources ouverts (voir ci-dessous).
Eléments prédéfinis
Eléments qui sont prédéfinis par PureBasic, comme les mots-clés, les fonctions ou les constantes.
Ajouter les éléments : du code source courant seulement
Les éléments ajoutés dans la liste de complétion sont seulement récupérés du code source courant.
Ajouter les éléments : du projet courant (si présent)
Les éléments ajoutés dans la liste de complétion sont récupérés à partir du projet courant, s'il y en a un. Les autres codes sources du projet n'ont pas à être obligatoirement ouverts dans l'IDE pour que ce soit fonctionnel.
Ajouter les éléments : du projet courant ou de tous les fichiers (si aucun)
Les éléments ajoutés dans la liste de complétion sont récupérés à partir du projet courant, s'il y en a un. Si le code source courant n'appartient pas au projet courant, tous les éléments des codes sources ouverts seront pris en compte.
Ajouter les éléments : de tous les fichiers ouverts
Les éléments ajoutés dans la liste de complétion sont récupérés à partir de tous les fichiers ouverts dans l'IDE.

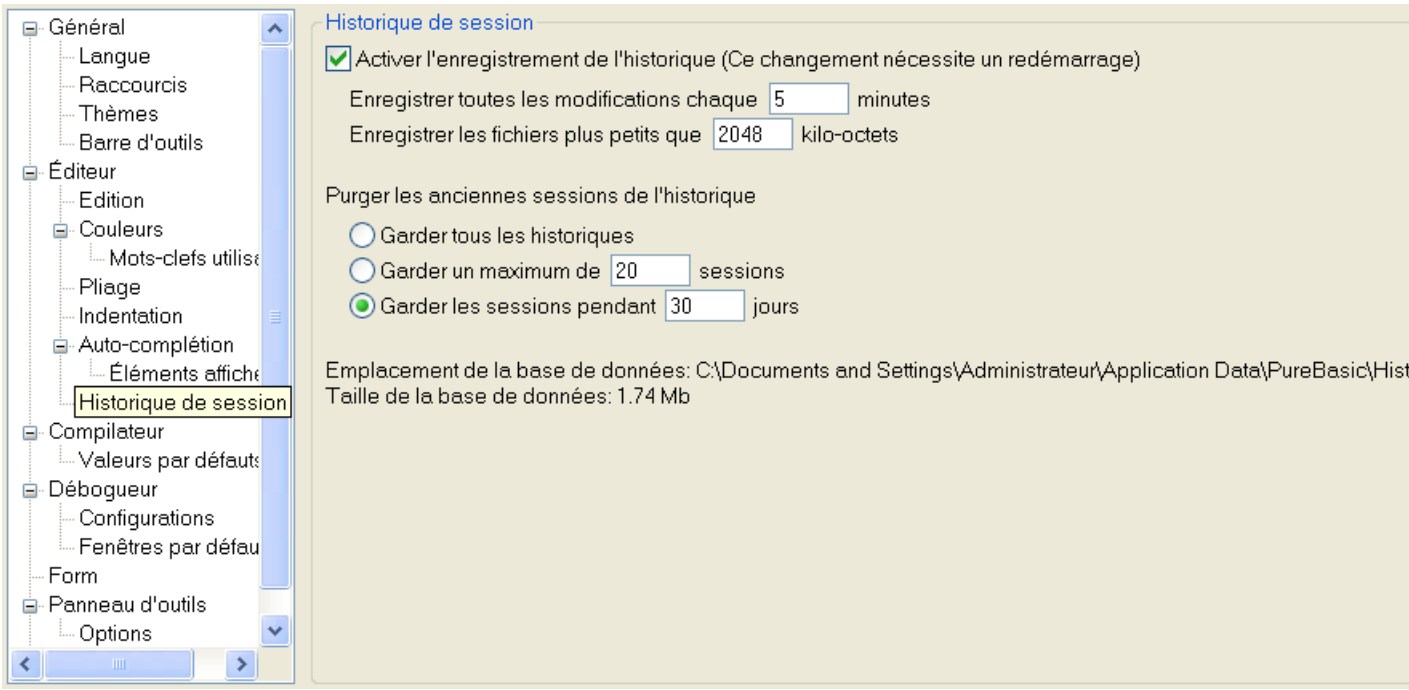
Editeur - Notes



Permet de configurer des marqueurs de 'Note' au niveau des commentaires du code source. Ces marqueurs peuvent être affichés dans les Notes du menu Outils ou dans l'outil de Explorateur de Procedure, et ils peuvent être marqués dans le code source avec un fond coloré.
La définition d'une Note :

Note
Un nom pour le type de Note.
Expression régulière
Une expression régulière définissant le motif de la Note. Cette expression régulière est appliquée à tous les commentaires dans le code source. Chaque match de l'expression est considéré en fonction du type de Note.
Priorité
A chaque type de Note est affecté une priorité. La priorité peut être utilisée pour ordonner et filtrer les Notes affichées.
Couleur
La couleur utilisée pour marquer la Note dans le code source (si activé). La couleur sera utilisée pour marquer le fond, soit du texte de la Note elle-même, soit de la ligne de code complète en fonction de l'option de coloration.
Afficher l'outil Note
Si activé, toutes les Notes trouvées sont répertoriées dans l'outil Note. Cette option peut être désactivée pour provoquer le marquage d'une Note avec une couleur de fond spéciale, si vous le souhaitez.
Afficher dans l'explorateur de procédure
Si activé, toutes les Notes trouvées sont présentées comme une entrée dans l'outil explorateur de procédure.

Editeur - Historique de Session



Permet de configurer la façon dont l'historique de la session enregistre les modifications.

Activer l'enregistrement de l'historique

Active ou désactive l'enregistrement de l'historique de la session. Lorsqu'il est activé, toutes les modifications apportées à un fichier seront enregistrées en tâche de fond, dans une base de données. Une session est créée lors du lancement de l'IDE, et est fermée lorsque l'IDE se ferme. Cette option est utile pour revenir à une version antérieure d'un fichier ou pour retrouver un fichier supprimé ou corrompu. C'est un peu comme un outil de sauvegarde des fichiers sources très puissant mais limité dans le temps (par défaut, un mois d'enregistrement). Il n'est pas destiné à remplacer un système de gestion des sources comme SVN ou GIT. Le code source sera stocké sans chiffrement, donc si vous travaillez sur code source sensible, assurez-vous d'avoir ce fichier de base de données dans un endroit sûr, ou de désactiver cette fonction. Il est possible de définir l'emplacement de la base de données en utilisant un commutateur en ligne de commande .

Enregistrer toutes les modifications chaque X minutes

Modifier l'intervalle entre chaque enregistrement automatique (lors de l'édition). Un fichier sera automatiquement enregistré lors de son enregistrement ou de sa fermeture.

Enregistrer les fichiers plus petits que X kilo-octets

Modifier la taille maximale (en kilo-octets) des fichiers en cours d'enregistrement. Ceci permet d'exclure les très gros fichiers qui pourraient rendre la base de données trop importante.

Garder tous les historiques

Gardez tout les historiques, la base de données n'est jamais purgée. Elle grossit toujours et devra donc être surveillée.

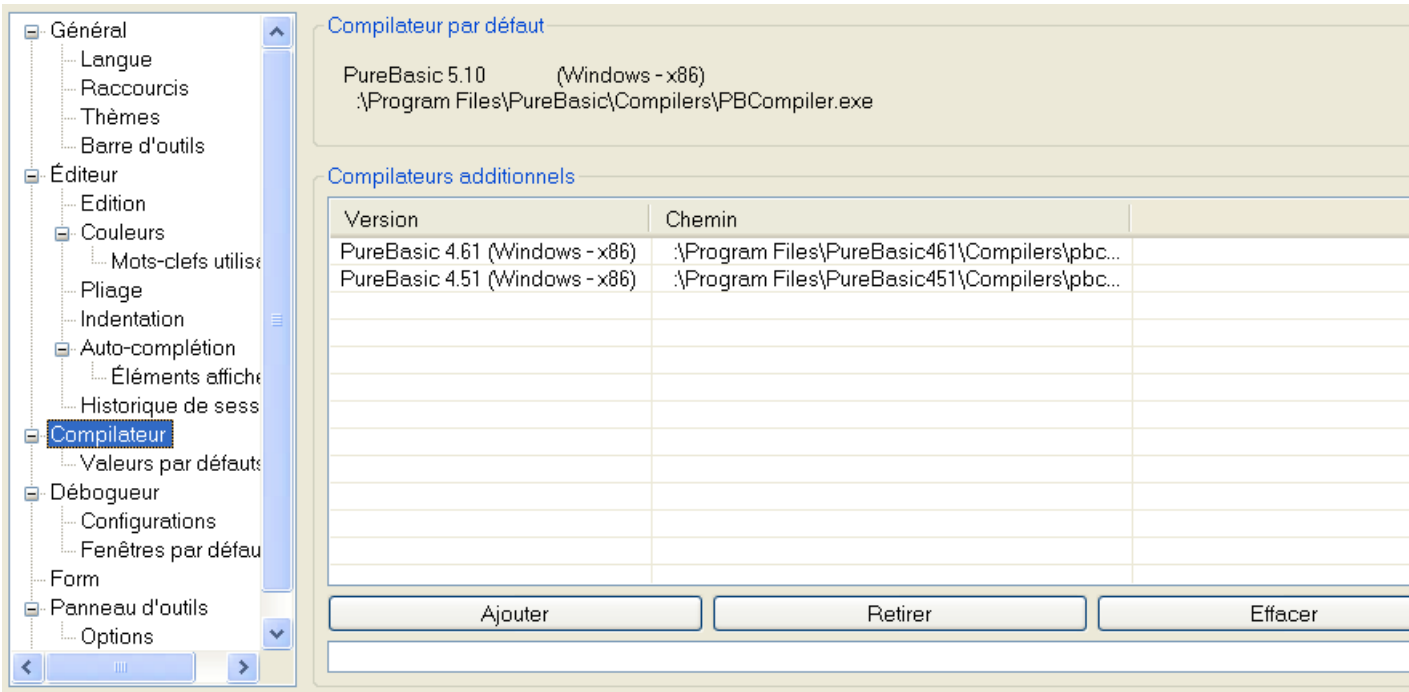
Garder un maximum de X sessions

Après avoir atteint le nombre maximal de sessions, la session la plus ancienne sera supprimée de la base de données.

Garder les sessions pendant X jours

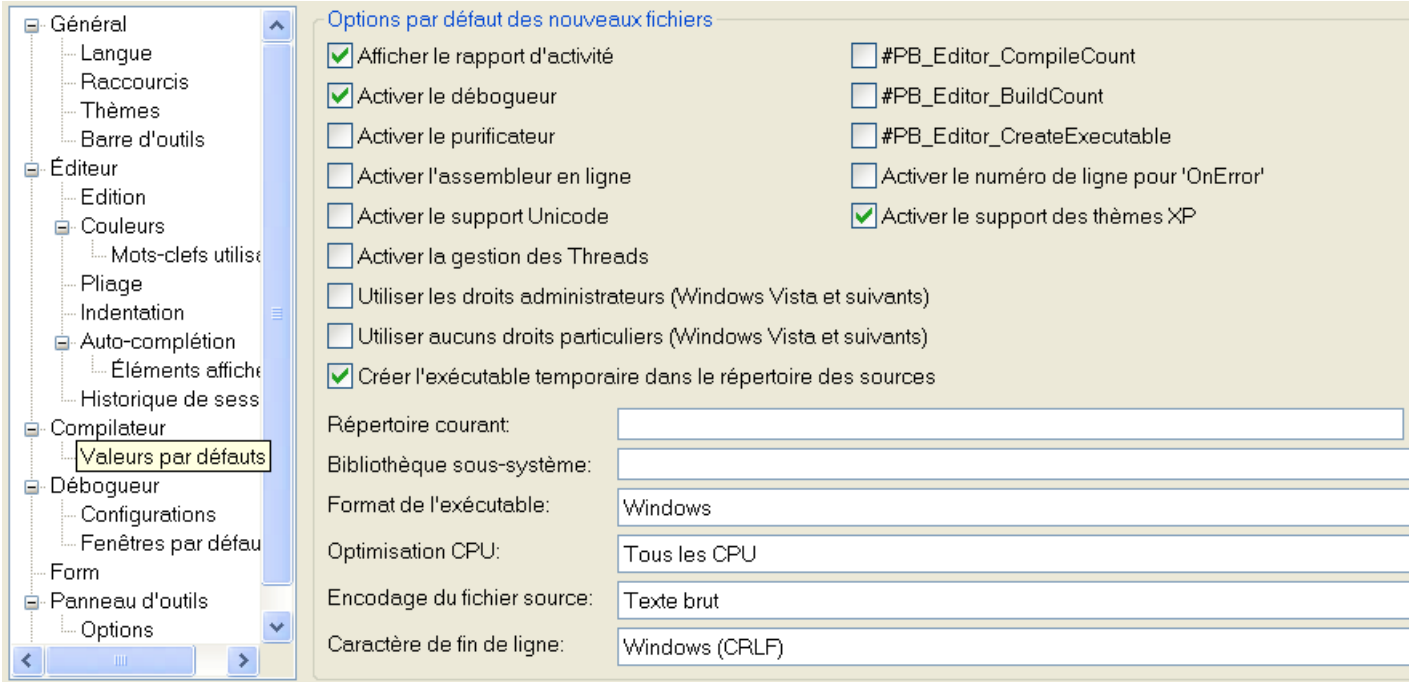
Après avoir atteint le nombre maximum de jours, la session sera supprimée de la base de données.

Compilateur



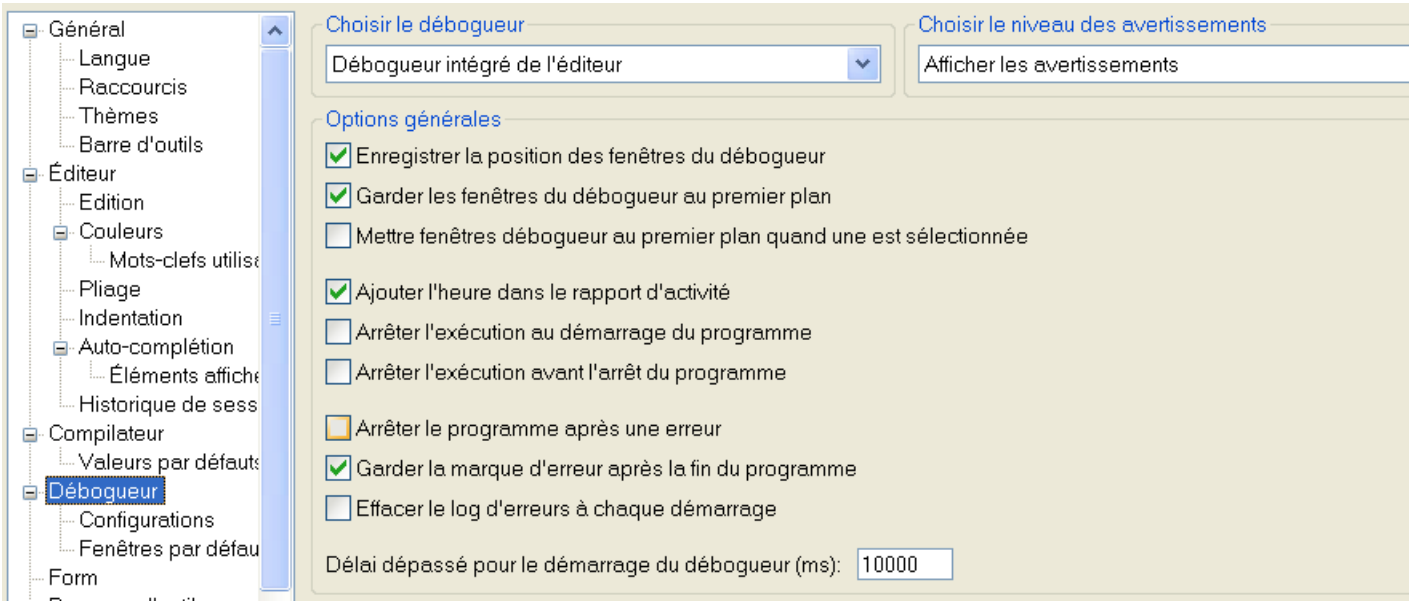
Cette section permet de configurer un compilateur additionnel qui sera ensuite disponible pour les options compilations . Ainsi, il est possible d'échanger rapidement le compilateur (x86 vers x64 par exemple) ou même d'utiliser d'anciens compilateurs directement dans le nouvel IDE.
N'importe quel compilateur à partir de la version 4.10 peut être ajouté ici. Le type de processeur supporté n'a pas à être le même que celui de l'IDE, tant que le système d'exploitation est identique. La liste affiche la version du compilateur et son chemin.
Les informations utilisées par l'IDE (pour la coloration syntaxique, l'autocomplétion, le visualisateur de structures) proviennent toujours du compilateur par défaut. Les compilateurs additionnels ne servent que pour la compilation.

Compilateur - Par Défauts



Cette page permet de régler les options par défaut qui seront utilisées lors de la création d'un nouveau fichier source à l'aide de l'IDE.
Pour une explication des paramètres, voir les options compilations .

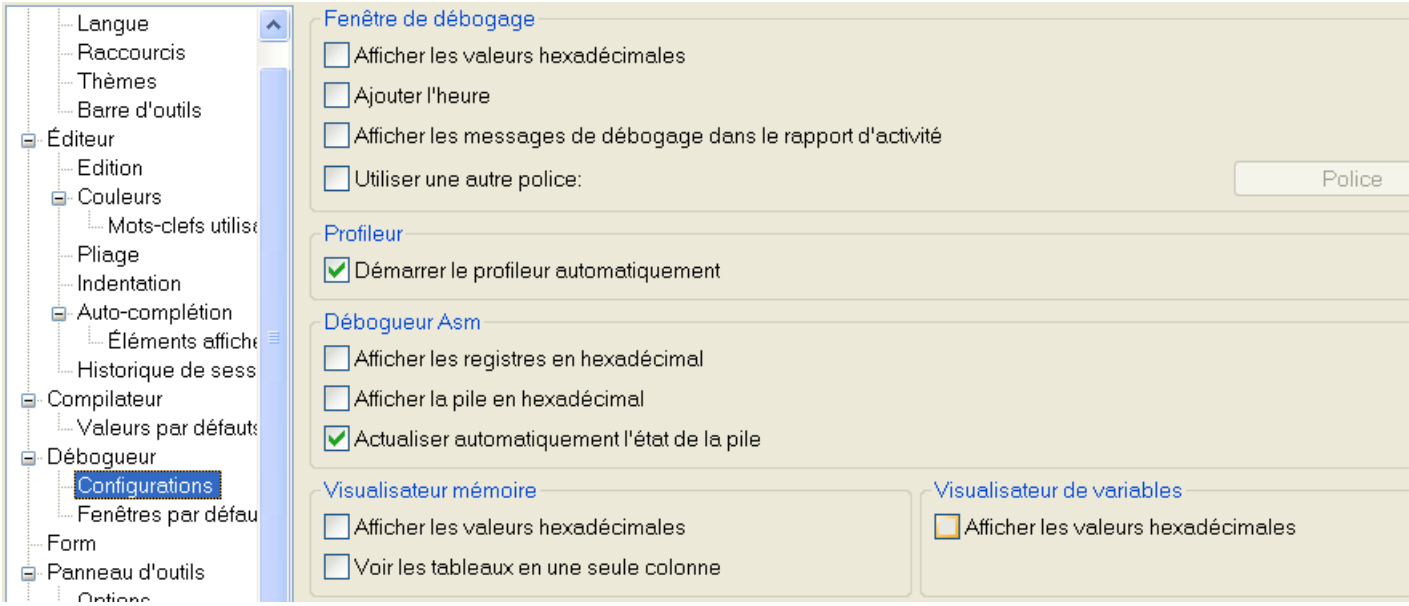
Débogueur



Permet de paramétrer le comportement du débogueur intégré ou indépendant. Le débogueur en ligne de commande ne se configure que par l'intermédiaire du terminal.
Type de débogueur
Sélectionne le type du débogueur à utiliser lorsqu'un programme est compilé dans l'IDE.
Enregistrer la position des fenêtres du débogueur
Même fonction que l'option "Mémoriser la position des fenêtres" de la section "Général", mais appliquée aux fenêtres du débogueur.
Garder les fenêtres du débogueur en premier plan
Toutes les fenêtres de débuggage seront toujours au premier plan, même devant les autres applications. Cela peut faciliter le débuggage des applications prenant toute la surface du bureau.
Mettre les fenêtres du débogueur au premier plan quand une est sélectionnée
Avec cette option activée, lorsqu'une fenêtre de débuggage est activée à l'aide de la souris ou du clavier, les autres fenêtres ouvertes sont automatiquement ramenées au premier plan.
Ajouter l'heure dans le rapport d'activité
Ajoute l'heure de chaque évènement dans le rapport d'activité.
Arrêter l'exécution au démarrage du programme
Chaque programme sera stoppé au début de son exécution, pour donner la possibilité de dérouler le programme pas à pas (sans avoir besoin de mettre un point d'arrêt au début de chaque source).
Arrêter l'exécution avant l'arrêt du programme
Chaque programme sera stoppé juste avant de quitter réellement. Cela permet par exemple d'utiliser les outils de débuggage pour examiner les variables ou la mémoire avant que le programme ne quitte.
Arrêter le programme après une erreur
Si le programme rencontre une erreur, il sera alors terminé automatiquement et toutes les fenêtres du débogueur seront fermées. Cela permet d'éditer le code immédiatement après une erreur, mais ce n'est plus possible d'examiner le programme pour essayer de déterminer la cause de l'erreur.
Garder les marqueurs d'erreurs après la fin du programme
N'efface pas les marqueurs d'erreurs quand le programme se termine. Cela permet de voir plus clairement ou l'erreur est survenue lors du retour à l'édition du code. Les marqueurs peuvent ensuite être enlevés à l'aide de la commande "Effacer les marqueurs d'erreurs" du sous-menu "Rapport d'activité".
Effacer le rapport d'activité à chaque exécution
Efface le rapport d'activité à chaque fois qu'un programme est exécuté. Cela assure que le rapport ne deviendra pas trop conséquent (cette option est aussi disponible quand le débogueur en ligne de

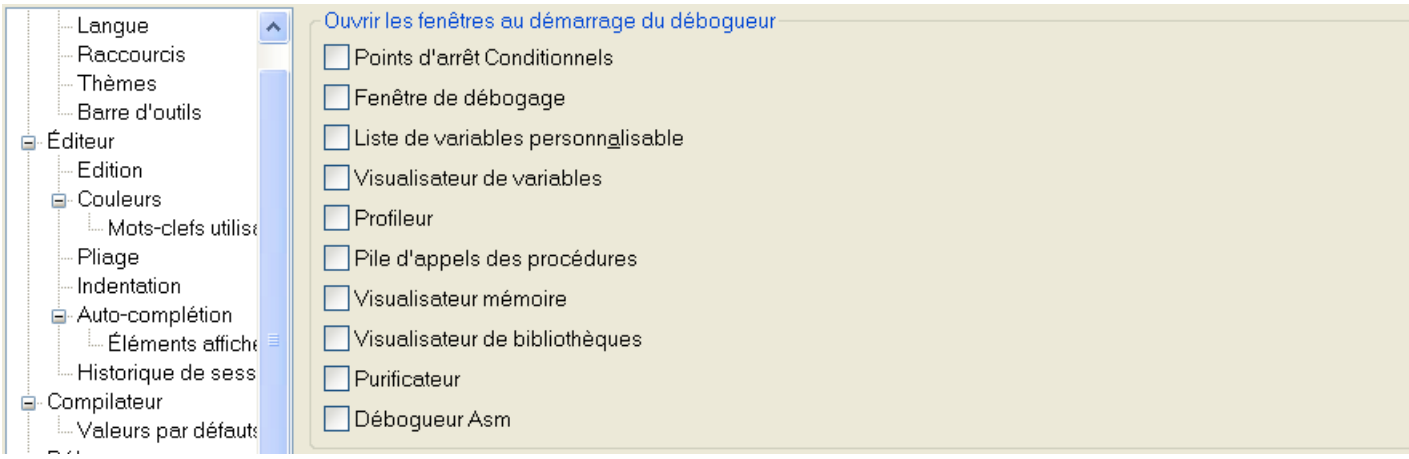
commande est sélectionné).
Delai maximum pour le lancement du débogueur
Indique le temps maximum, en millisecondes, qui sera donné au programme pour s'initialiser. Ce delai évite que le débogueur ne bloque indéfiniment si l'exécutable à déboguer ne peut pas se lancer, pour une raison ou pour une autre.

Débogueur - Configuration des outils



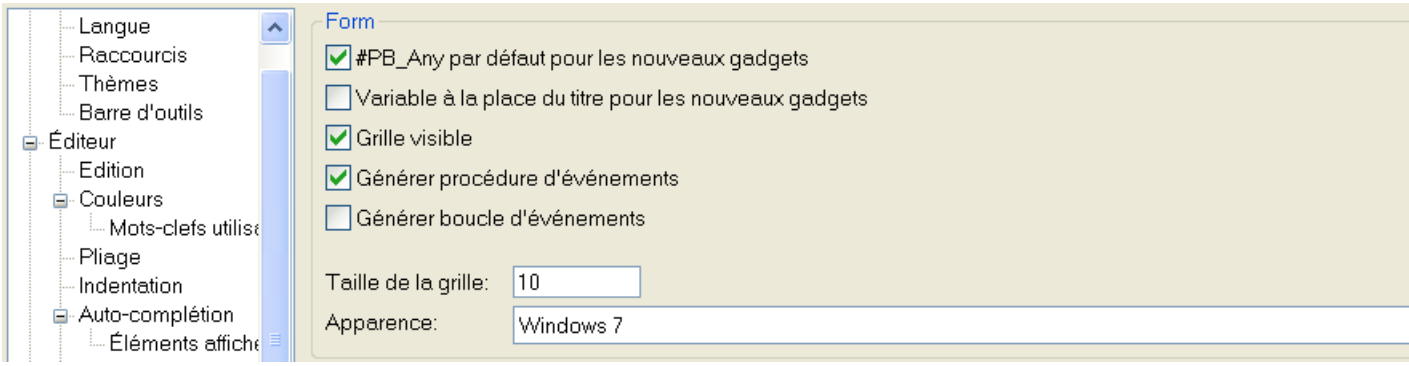
Cela permet de contrôler l'affichage et le comportement des outils intégrés au débogueur. Les options "Afficher les valeurs en hexadecimal" change l'affichage des octets, word ou long du format décimal (base 10) au format hexadécimal (base 16).
Fenêtre de débogage Ajouter l'heure
Ajoute l'heure à chaque ligne affichée dans la fenêtre de débogage.
Fenêtre de débogage - Affiche les messages de débogage dans le rapport d'activité
Avec cette option activée, une commande Debug dans le code n'ouvrira pas la fenêtre du débogueur , mais affichera les messages de débogage dans le rapport d'activité .
Fenêtre de débogage Utilise une autre police
Une police personnalisée peut être sélectionnée pour l'affichage dans la fenêtre de débogage. Ce qui permet de choisir une petite police pour afficher plus de données ou de choisir une police avec une taille proportionnelle quand c'est nécessaire.
Profileur - Lancer le profileur au démarrage du programme
Détermine si le profileur commencera l'enregistrement des données au démarrage du programme.
Débogueur ASM Actualiser automatiquement l'état de la pile
Met à jour automatiquement l'état de la pile à chaque "Pas" ou "Stop" effectué. Si cette option est désactivée, un bouton "Actualiser" sera affiché dans la fenêtre ASM.
Observateur de mémoire Voir les tableaux en une seule colonne
Si la zone de mémoire est affichée en mode "tableau", cette option indique si ce tableau sera multi-colonnes (avec 16 octets par colonne) ou seulement avec une colonne.

Débogueur - Fenêtres par défaut



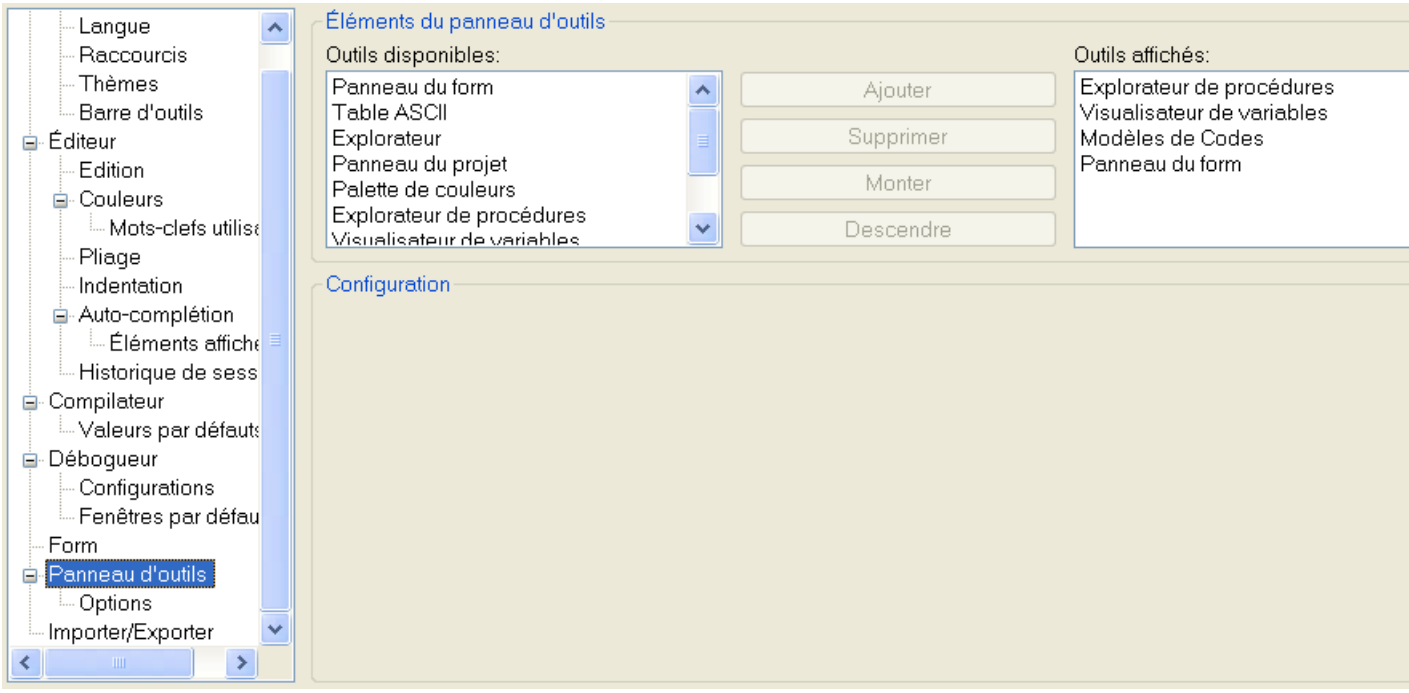
Les fenêtres cochées sur cette page seront ouvertes automatiquement à chaque fois qu'un programme sera lancé avec le débogueur.

Form



Permet de personnaliser le comportement du concepteur de fenêtre (Form) intégré. **#PB_Any par défaut pour les nouveaux gadgets**
Si elle est activée, la création du nouveau gadget utilisera '#PB_Any' au lieu d'une numéro.
Variable à la place du titre pour les nouveaux gadgets
Si elle est activée, le nouveau gadget utilisera une variable au lieu d'un texte de titre prédéfini. Cela peut être utile pour localiser facilement la fenêtre.
Grille visible
Si elle est activée, la grille sera visible sur le concepteur de fenêtres, afin de faciliter l'alignement des gadgets.
Générer procédure d'événements
Si elle est activée, une procédure d'événement sera générée automatiquement (et à mise jour).
Générer boucle d'événements
Si elle est activée, une boucle d'événement de base est générée pour le form.
Taille de la grille
L'espace entre les deux points de la grille, en pixels.
Apparence
Habillage à utiliser pour le concepteur de forms.

Panneau d'outils



Gère la configuration des outils internes qui peuvent être affichés dans le panneau latéral. Chaque outil qui est dans la liste des "Outils affichés" sera affiché dans la palette d'outils. Les outils qui ne sont pas présents dans cette liste sont accessibles à partir du menu "Outils". Ils seront ouverts dans une fenêtre indépendante.
Il est préférable de ne mettre que les outils qui sont utilisés très fréquemment dans la palette d'outils, et de mettre le plus utilisé en premier, car c'est celui qui est affiché le premier lorsque l'IDE démarre.
En sélectionnant un outil dans l'une des deux listes, un panneau de configuration spécifique à cet outil (si il y en a un) apparaîtra dans la partie "Configuration".
Les outils suivants peuvent être configurés :

Exploreur

Il est possible de choisir entre un affichage en mode liste ou en mode hierarchique (arbre) et de mémoriser ou non le dernier répertoire affiché (s'il n'est pas mémorisé, alors le "Répertoire par défaut" défini dans les options "Général" sera utilisé)

Navigateur de procédures

"Trier les procédures par nom" : trie la liste des procédures par ordre alphabétique (par défaut, les procédures apparaissent dans l'ordre de leurs déclarations)
"Grouper les marqueurs" : groupe les marqueurs (";-") entre eux.
"Afficher les arguments des procédures" : Affiche le nom de la procédure et tous ses paramètres.

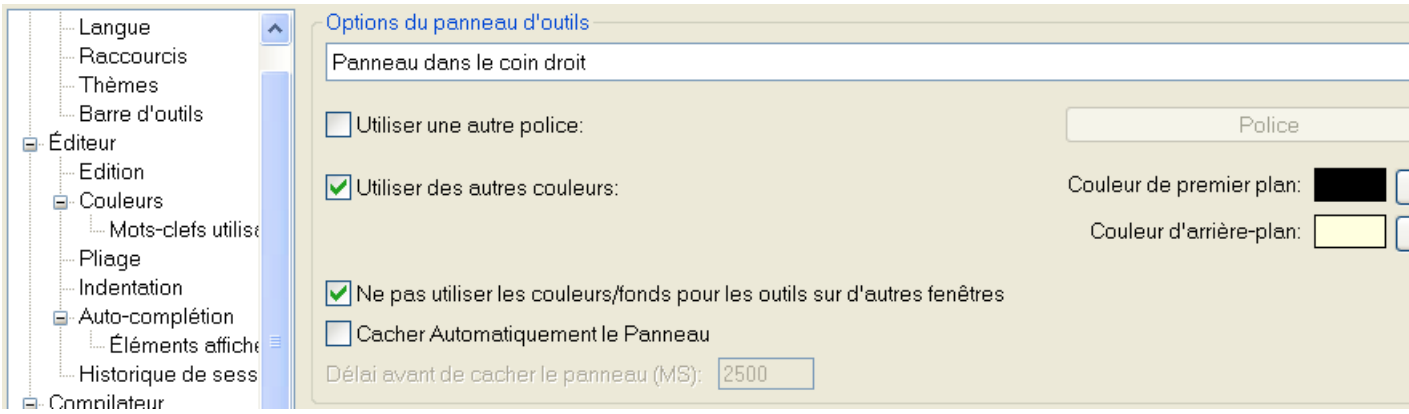
Visualisateur de variables

L'option "Afficher les éléments de tous les fichiers ouverts" détermine si la liste doit contenir les éléments du fichier en cours d'édition, ou de tous les fichiers actuellement ouverts par l'IDE.
De plus, il est possible de sélectionner le type d'éléments qui seront affichés dans le visualisateur de variables.

Outils Aide

Appuyer sur F1 ouvre l'aide dans le panneau : Spécifie s'il faut ouvrir l'outil d'aide au lieu du visualisateur d'aide lorsque F1 est pressé.

Panneau d'outils - Options



L'apparence de la palette d'outil en elle même peut être configurée dans cette section : le côté qui sera utilisé pour afficher la palette (à droite ou à gauche de la zone d'édition), la police de caractères, ainsi que la couleur de fond et d'avant plan utilisée par les outils affichés. La police et les couleurs optionnelles peuvent être désactivées pour utiliser les valeurs par défaut de l'OS.

Ignorer la couleur/police dans les fenêtres indépendantes

Si cette option est activée, les options de couleur/police ne seront appliquées que si l'outil est affiché dans la palette d'outils (ceux ouverts via le menu "Outils" utiliseront la couleur/police par défaut).

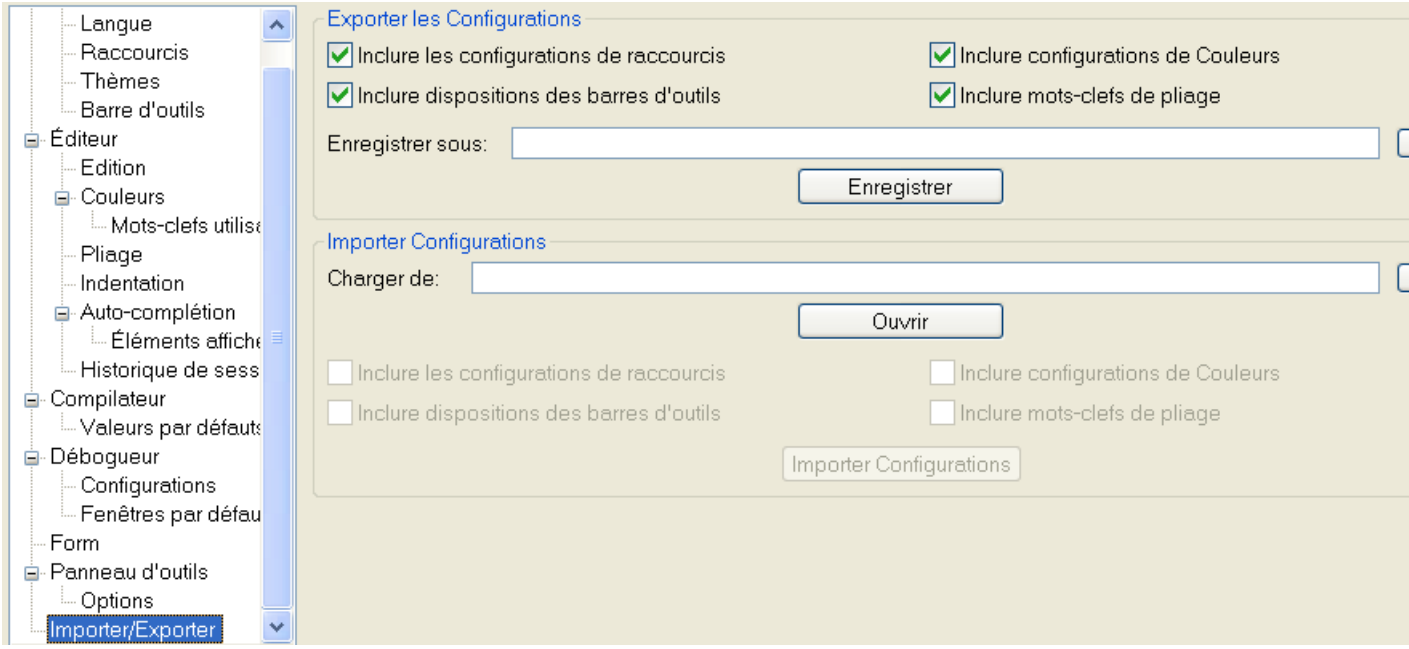
Cacher automatiquement la palette d'outils

Pour gagner un peu de place, la palette d'outils peut se cacher automatiquement si la souris ne se trouve pas au dessus d'elle. Pour la faire réapparaître, il suffit de déplacer la souris sur la zone située sur le côté de la zone d'édition.

Délai en millisecondes avant de cacher la palette

Défini un délai (en ms), après lequel la palette d'outils sera cachée si la souris n'est plus au dessus d'elle.

Importer/Exporter



Il est possible d'importer et d'exporter la configuration de l'IDE dans un format indépendant, et de l'importer dans un autre IDE (sous un autre OS par exemple, ou pour le partager avec d'autres utilisateurs).

Pour réaliser une exportation, il faut sélectionner le type des paramètres à exporter, saisir un nom de fichier et cliquer sur le bouton "Enregistrer"

Pour importer des paramètres, il suffit de choisir le nom du fichier et de cliquer sur "Ouvrir". Les paramètres disponibles seront alors affichés, il faudra décocher les paramètres indésirables puis cliquer sur "Importer".
Pour que les nouveaux paramètres prennent effet, il faudra appuyer sur le bouton "Appliquer".
Note : il est possible d'importer les fichiers 'styles' de l'éditeur jaPBe, mais seuls les paramètres concernant les couleurs seront importés.

Chapitre 19

Commutateurs de la ligne de commande

L'IDE de PureBasic permet de modifier les chemins et les fichiers utilisés lors de son lancement. Cela permet de créer plusieurs raccourcis qui vont lancer l'IDE avec une configuration particulière pour chaque utilisateur, ou pour chaque projet.
Il existe également des options pour la compilation des projets PureBasic directement en ligne de commande. Construire un projet à partir de la ligne de commande comporte les mêmes actions, comme le choix de la 'Cible' ou "Créer toutes les cibles" du menu Compilateur .
Options générales :

```
/VERSION                  affiche la version de l'IDE
/HELP ou /?               affiche une description des arguments de la
 ligne de commande

/P <fichier Préférences>             charge (et enregistre) la
 configuration à partir du fichier spécifié.
/T <fichier Modèle>                  charge (et enregistre) les
 modèles de code à partir du fichier spécifié.
/A <fichier outils>                  charge (et enregistre) la
 configuration des outils externes à partir du fichier spécifié.
/S <Chemin Source>                   change l'option "Répertoire
 principal" des préférences.
/E <Chemin Explorateur>              démarre l'outil 'Explorateur'
 avec le chemin spécifié.
/L <Numéro ligne>                    déplace le curseur à la ligne
 indiquée (seulement dans le dernier fichier chargé).
/H <Base de données de l'Historique> le fichier à utiliser pour la
 base de données de l'historique de la session.
/NOEXT                               désactive l'association
 automatique des fichiers '.pb' dans la base de registre.
/LOCAL                               place tous les fichiers
 préférences dans le répertoire de PureBasic au lieu du répertoire
 personnel de l'utilisateur.
/PORTABLE                            combinaison de /LOCAL et de
 /NOEXT
```

Options pour compiler les projets :

```
/BUILD <file>             spécifie le fichier de projet à construire
/TARGET <target>          spécifie la cible à construire (la valeur
 par défaut est de construire toutes les cibles)
/QUIET                    cache tous les messages lors de la
 construction sauf les erreurs
/READONLY                 ne met pas à jour le fichier de projet après
 la compilation (ni nouvelle date d'accès et ni nouveaux compteurs de
```

```
    construction )
```

Par défaut, les fichiers paramètres utilisés avec /P /T et /A se trouvent dans le répertoire %APPDATA%\PureBasic\.

```
  Le paramètre /NOEXT est particulièrement utile lorque plusieurs
   versions de PureBasic
  cohabitent sur un même système (pour tester les versions bêta par
   exemple). Ainsi les
  fichiers .pb seront toujours associés à la même version de PureBasic.
\\



  Le paramètre /PORTABLE peut être utilisé afin de garder l'ensemble de
   la configuration
  dans le répertoire local pour copier facilement PureBasic sur
   différents ordinateurs
  (ou le faire fonctionner depuis une clé USB par exemple).
```

Exemple :

```
1   PureBasic.exe Example.pb /PORTABLE
```

Il est aussi possible d'ouvrir des fichiers en ligne de commande, par exemple la commande : PureBasic.exe Exemple.pb, ouvrira le fichier Exemple.pb dans l'éditeur de PureBasic. Il est même possible de spécifier un masque en entrée, par exemple "*.pb" chargera tous les fichiers sources du répertoire.

Troisième partie

La langue

Chapitre 20

Travailler avec différentes bases numériques

(Note : Ces exemples utilisent le symbole ^ pour signifier 'élevé à la puissance de' - ceci est seulement une convention dans ce document, PureBasic n'a pas actuellement d'opérateur d'élévation à la puissance ! Utilisez plutôt la commande PureBasic Pow() de la bibliothèque "Math".)

Introduction

Une base numérique est une manière de représenter les nombres, en utilisant une certaine quantité de symboles possibles par chiffre. La plus courante que vous devez connaître est la base 10 (c-a-d décimale), car il y a 10 chiffres utilisés (0 à 9), et c'est celle que la plupart des humains peuvent manipuler le plus facilement.
Le but de cette page est d'expliquer différentes bases numériques et comment vous pouvez travailler avec elles. En effet les ordinateurs travaillent en binaire (base 2) et il y a certains cas où il est avantageux de le comprendre (par exemple, lors de l'utilisation d'opérateurs logiques, masques de bits, etc).

Vue d'ensemble des bases numériques

Le système Décimal

Pensez à un nombre décimal, et réfléchissez à sa représentation en colonnes. Prenons par exemple le nombre 1234. Séparé en colonnes, nous avons :

```
1   2   3   4
```

Réfléchissez à ce que sont les en-têtes de chaque colonne. Nous savons qu'il y a les unités, les dizaines, les centaines et les milliers, présentés ainsi :

```
1000  100   10    1
   1    2    3    4
```

Nous pouvons voir que le nombre 1234 est constitué de

```
   1*1000=1000
+ 2* 100= 200
+ 3*  10=  30
+ 4*   1=   4
Total   =1234
```

Si nous regardons également les en-têtes de colonne, vous verrez qu'à chaque fois que vous bougez d'une colonne vers la gauche nous multiplions par 10, qui justement s'avère être la base numérique. Chaque fois que vous bougez d'une colonne vers la droite, vous divisez par 10. Les en-têtes de colonnes peuvent être appelées les poids, puisque pour obtenir tout le nombre nous devons multiplier les chiffres de chaque colonne par le poids. Nous pouvons exprimer les poids en utilisant des index. Par exemple 10 ^2 signifie '10 élevé à la puissance de 2' ou 1*10*10 (=100). De même, 10 ^4 signifie 1*10*10*10*10 (=10000). Remarquez dans ces exemples, que peu importe la valeur de l'index, il correspond au nombre de fois que nous multiplions le nombre qui doit être élevé. 10 ^0 signifie 1 (puisque nous multiplions par 10 aucune fois). Utiliser des nombres négatifs montre que nous devons diviser, par exemple 10 ^-2 signifie 1/10/10 (=0.01). Les valeurs de l'index prennent plus de sens quand nous attribuons un numéro à chaque colonne - vous verrez souvent des choses comme 'bit 0' qui signifie en fait 'chiffre binaire en colonne 0'.

```
Dans cet exemple, ^ signifie élevé à la puissance de, donc 10^2
 signifie 10 élevé à la puissance de 2.
Numéro de colonne          3     2     1     0
Poids (index)          10^3  10^2  10^1  10^0
Poids (valeur réelle)  1000   100    10     1
Nombre exemple (1234)     1     2     3     4
```

Précédemment nous avons vu comment convertir le nombre 1234 dans son équivalent décimal. Conversion peu justifiée, car il était déjà en décimal mais nous pouvons en tirer la méthode générale - voici donc comment convertir n'importe quel nombre en sa valeur décimale :

```
B = Valeur de la base numérique

1) Séparer le nombre, peu importe la base, en colonnes. Par
   exemple, si nous avions la valeur 'abcde' dans notre base
 numérique fictive 'B',
   les colonnes seraient : a   b   c   d   e

2) Multiplier chaque symbole par le poids de chaque colonne (le poids
 étant
   calculé par 'B' élevé à la puissance du numéro de la colonne):
     a * B^4 = a * B * B * B * B
     b * B^3 = b * B * B * B
     c * B^2 = c * B * B
     d * B^1 = d * B
     e * B^0 = e

3) Calculer la somme de toutes ces valeurs. En écrivant toutes ces
 valeurs dans leur
   équivalent décimal pendant les calculs, il devient beaucoup plus
 facile de voir le
   résultat and de faire le calcul (si nous convertissons en décimal).
```

Convertir dans la direction opposée (de décimal vers la base numérique 'B') s'effectue en utilisant la division au lieu de la multiplication :

```
1) Commencer par le nombre décimal que vous voulez convertir (e.g.
 1234).

2) Diviser par la base numérique ciblée ('B') et prendre note du
 quotient et du
   reste.

3) Diviser le quotient de (2) par la base numérique ciblée ('B') et
 prendre note du
   quotient et du reste.

4) Continuer à diviser comme ça jusqu'à ce que vous obteniez un
 quotient de 0.
```

```
5) Votre nombre dans la base numérique ciblée est constitué des
 restes écrits dans l'ordre
   du dernier calculé au premier calculé. Par exemple, votre nombre
 serait constitué
   des restes des étapes dans cet ordre 432.
```

Des exemples plus particuliers seront donnés dans les paragraphes concernant les bases numériques particulières.

Système binaire

Tout dans un ordinateur est stocké en binaire (base 2, avec les symboles définis '0' ou '1') mais le travail avec des nombres binaires suit les mêmes règles qu'en décimal. Chaque symbole dans un nombre binaire est appelé bit, abréviation de BInary digiT (chiffre binaire). Généralement, vous travaillerez avec des bytes (octets, 8-bit), words (mots, 16-bit) ou longwords (mots longs, 32-bit) car ce sont les tailles par défaut des types intégrés à PureBasic. Les poids pour un byte (octet) sont les suivants :

```
(^ signifie 'élevé à la puissance de', la base numérique est 2 pour
 le binaire)
Bit/numéro de colonne      7     6     5     4     3     2     1     0
Poids (index)            2^7   2^6   2^5   2^4   2^3   2^2   2^1   2^0
Poids (valeur réelle)    128    64    32    16     8     4     2     1
```

Donc, par exemple, si nous avions le nombre 00110011 (Base 2), nous pourrions établir sa valeur comme ceci :

```
  0 * 128
+ 0 *  64
+ 1 *  32
+ 1 *  16
+ 0 *   8
+ 0 *   4
+ 1 *   2
+ 1 *   1
=      51
```

Un exemple de conversion inversée serait d'écrire la valeur 69 en binaire. Nous ferions comme ceci :

```
69 / 2 = 34 r 1     ^
34 / 2 = 17 r 0    /|\
17 / 2 =  8 r 1     |
 8 / 2 =  4 r 0     |    Lire les restes dans cette direction
 4 / 2 =  2 r 0     |
 2 / 2 =  1 r 0     |
 1 / 2 =  0 r 1     |
(Arrêt ici car le quotient de la dernière division était 0)

Lire les restes de bas en haut pour obtenir la valeur en binaire =
  1000101
```

Une autre chose à considérer lorsque l'on travaille avec des nombres binaires est la représentation des nombres négatifs. Comme nous le faisons quotidiennement, nous pourrions simplement mettre un symbole négatif devant le nombre décimal. Nous ne pouvons pas faire cela en binaire, mais il y a un moyen (après tout, PureBasic travaille principalement avec des nombres signés, et doit donc être capable de gérer les nombres négatifs). Cette méthode est appelée 'complément à deux' et indépendamment de toutes les bonnes fonctionnalités que cette méthode possède (et qui ne seront pas expliquées ici, pour épargner une certaine confusion) la manière la plus simple de la comprendre, est de se dire que le poids

du bit le plus significatif (Most Significant bit / le MSb est le numéro du bit avec la plus grande valeur. Dans le cas d'un octet (byte), ce serait le bit 7) est réellement une valeur négative. Donc pour un système de complément à deux, les poids des bits sont modifés en :

```
(^ signifie 'élevé à la puissance de', la base numérique est 2 pour
 le binaire)
Bit/numéro de colonne      7      6      5      4      3      2      1      0
Poids (index)           -2^7   2^6    2^5    2^4    2^3    2^2    2^1    2^0
Poids (valeur réelle)   -128    64     32     16      8      4      2      1
```

et vous pourrez faire la conversion de binaire vers décimal exactement de la même manière que ci-dessus, mais en utilisant le nouveau jeu de poids. Par exemple, le nombre 10000000 (Base 2) est -128, et 10101010 (Base 2) est -86.
Pour convertir un nombre binaire positif en négatif et vice versa, vous inversez tous les bits et ensuite ajoutez 1. Par exemple, 00100010 passerait en négatif en inversant -> 11011101 et additionant 1 -> 11011110.
Ceci facilite la conversion de décimal à binaire, car vous pouvez convertir les nombres décimaux négatifs comme leurs équivalents positifs (en utilisant la méthode ci-dessus) et ensuite rendre le nombre binaire négatif à la fin.
Les nombre binaires dans PureBasic sont précédés du symbol 'pourcentage', et évidement tous les bits du nombre doivent être un '0' ou un '1'. Par exemple, vous pouvez utiliser la valeur %110011 dans PureBasic pour signifier 51. Notez que vous n'avez pas besoin de mettre les '0' de tête (ce nombre serait vraiment %00110011) mais cela peut faciliter la lisibilité de votre source si vous écrivez l'ensemble des bits.

Le système Hexadécimal

L'hexadecimal (pour base 16, symboles '0'-'9' suivis de 'A'-'F') est une base numérique qui est plus généralement employée en informatique, car il s'agit probablement de la base numérique non décimale la plus facile à comprendre pour les humains, et vous n'avez pas à écrire de longues chaines de symboles pour vos nombres (comme c'est le cas avec le binaire).
Les mathématiques hexadécimales suivent les mêmes règles qu'en décimal, bien que vous ayez maintenant 16 symboles par colonne jusqu'à ce que vous ayez besoin d'une retenue. La conversion entre l'hexadécimal et le décimal suit les mêmes principes qu'entre le binaire et le décimal, sauf que les poids sont des multiples de 16 et les divisions sont faites par 16 au lieu de 2 :

```
Numéro de colonne          3      2      1      0
Poids (index)           16^3   16^2   16^1   16^0
Poids (valeur réelle)   4096    256     16      1
```

Convertir la valeur hexadécimale BEEF (Base 16) en décimal serait fait comme ceci :

```
  B * 4096 = 11 * 4096
+ E *  256 = 14 *  256
+ E *   16 = 14 *   16
+ F *    1 = 15 *    1
=               48879
```

Et convertir la valeur 666 en hexadécimal serait fait comme ceci :

```
666 / 16 = 41 r 10    ^
 41 / 16 =  2 r  9   /|\    Lire les chiffres dans cette direction,
 se souvenir de convertir
  2 / 16 =  0 r  2    |     en chiffres hexadecimaux où nécessaire
(Arrêt ici car le quotient de la dernière division était 0)
La valeur hexadécimale de 666 est 29A
```

La chose vraiment intéressante avec l'hexadécimal c'est qu'il permet également de convertir vers le binaire très facilement. Chaque chiffre hexadécimal représente 4 bits, pour convertir entre hexadécimal et binaire, vous devez juste convertir chaque chiffre hexadécimal en 4 bits ou chaque groupe de 4 bits vers

un chiffre hexadécimal (en se rappelant que 4 est un diviseur de toutes les tailles communes de nombres binaires dans un CPU). Par exemple :

```
Nombre hexadécimal      5         9         D         F           4E
Valeur binaire       0101      1001      1101      1111        01001110
```

Lorsque vous utilisez des valeurs hexadécimales dans PureBasic, vous mettez un symbole 'dollar' devant le nombre, par exemple $BEEF.

Les conversions Hexadécimal <-> Binaire

Il est aisé de convertir un nombre hexadécimal en un nombre binaire et inversement, en utilisant les 'nibles'. Un nible est un groupe de 4 bits, appelé demi octet ou quartet. Un quartet contenant 4 bits, il peut donc prendre seize (2 ^4) valeurs différentes et correspond donc à un chiffre en hexadécimal. Deux chiffres hexadécimaux formant un octet, ce dernier est souvent représenté par deux nibbles.
Par exemple :

```
Numéro de colonne          3       2       1       0
Poids (index)            2^3     2^2     2^1     2^0
Poids (valeur réelle)      8       4       2       1
                          -------------------------
                  $7       0       1       1       1  : 0*8 + 1*4 + 1*2
 + 1*1 = 7
                  $C       1       1       0       0  : 1*8 + 1*4 = 12 =
 $C
```

Comment s'écrit $F2 en binaire ?

```
Poids (valeur réelle)      8   4   2   1 |  8   4   2   1
                          -------------|-------------
                  $F2      1   1   1   1    0   0   1   0 : 1*8 + 1*4 + 1*2
 + 1*1 = 15 = $F
                                                    et 0*8 + 0*4 + 1*2
 + 0*1 =  2 = $2
```

Facile non !

Chapitre 21

Break : Continue

Syntax

```
Break [Niveau]
```

Description

Break permet de quitter à n'importe quel moment une ou plusieurs des boucles suivantes : Repeat , For , ForEach et While .

Arguments

Niveau (optionnel) Indique le nombre de boucles qui doivent être abrégées.
Sans paramètre, Break quitte la boucle en cours.

Valeur de retour

Aucune.

Exemple : Boucle simple

```
1  For k=0 To 10
2    If k=5
3      Break  ; Quitte immédiatement la boucle For/Next
4    EndIf
5    Debug k
6  Next
```

Exemple : Boucles imbriquées

```
1  For k=0 To 10
2    Counter = 0
3    Repeat
4      If k=5
5        Break 2 ; Quitte immédiatement les boucles Repeat/Until et
   For/Next
```

```
6       EndIf
7       Counter+1
8     Until Counter > 1
9     Debug k
10  Next
```

Syntax

```
Continue
```

Description

Continue permet de passer directement à la prochaine itération dans l'une des boucles suivantes : Repeat , For , ForEach et While .

Arguments

Aucun.

Valeur de retour

Aucune.

Exemple

```
1   For k=0 To 10
2     If k=5
3       Continue  ; N'affichera pas 'Debug 5' car la fin de cette
      itération sera ignorée
4     EndIf
5     Debug k
6   Next
```

Chapitre 22

Utiliser le compilateur en ligne de commande

Introduction

Le compilateur est situé dans le sous-dossier 'compilers' du dossier PureBasic. La manière la plus simple d'y accéder est d'ajouter ce répertoire à la variable PATH, ce qui vous donnera un accès permanent à toutes les commandes de ce répertoire. Chaque commutateur de commande qui commence par '/' ne concerne que Windows.

Paramètres du compilateur (multi-plateforme)

-h, –help, /? : affiche une aide simplifiée sur le compilateur.
-c, –commented, /COMMENTED : crée un fichier de sortie '.asm' commenté en même temps que l'exécutable. Ce fichier peut être ré-assemblé ultérieurement après l'avoir modifié selon vos besoins. Cette option est à destination des programmeurs avancés.
-d, –debugger, /DEBUGGER : active le débogueur.
-pf, –purifier, /PURIFIER : active le purifier. Le debugger doit être activé pour faire fonctionner le purifier.
-e, –executable, /EXE "Fichier" : crée un fichier exécutable ou une DLL indépendant ayant pour nom 'Fichier' et enregistré dans le chemin désiré. Sur MacOS X, il est possible de créer un ensemble d'applications en ajoutant ".app" dans le nom de l'exécutable. De cette façon, toute la structure de répertoire sera créé automatiquement.
-r, –resident, /RESIDENT "Fichier" : crée un fichier résident ayant pour nom 'Fichier'.
-i, –import, /IMPORT "Fichier" : crée un fichier d'importation au nom de fichier donné. Seulement un seul bloc Import/EndImport autorisé dans le fichier. Les fonctions importées seront chargées automatiquement pour tous les projets de PureBasic.
-l, –linenumbering, /LINENUMBERING : ajoute des informations concernant les lignes et les fichiers sources à l'exécutable, qui peut le ralentir considérablement. Ce qui permet à la bibliothèque OnError d'indiquer le fichier et le numéro de ligne en cas d'erreur.
-q, –quiet, /QUIET : désactive toutes les éditions de texte inutiles. Très pratique lorsque vous utilisez un éditeur tiers. -sb, –standby, /STANDBY : charge le compilateur en mémoire, en attente de commandes externes (éditeurs, scripts...). Plus d'informations sur l'utilisation de ce paramètre est disponible dans le fichier 'CompilerInterface.txt' du répertoire 'SDK' de PureBasic.
-ir, –ignoreresident, /IGNORERESIDENT "Fichier" : ne charge pas le fichier résident au démarrage du compilateur. C'est utile pour mettre à jour un fichier résident déjà existant.
-o, –constant, /CONSTANT Nom=Valeur : crée la constante spécifiée avec la valeur indiquée. Exemple : 'pbcompiler test.pb /CONSTANT MaConstante=10'. La constante #MaConstante sera créée automatiquement avec la valeur 10 avant de commencer la compilation.
-u, –unicode, /UNICODE : utilise l' unicode en lieu et place de l'ASCII pour la gestion des chaînes de

caractères.
-t, –thread, /THREAD : utilise les routines thread-safe pour toutes la gestion des chaînes de caractères et pour toutes les commandes.
-s, –subsystem, /SUBSYSTEM "Nom" : utilise le sous-système spécifié pour remplacer des commandes internes. Pour plus d'informations, voir sous-systèmes . -k, –check, /CHECK : vérifie la syntaxe uniquement, ne créer pas ni ne lance l'exécutable.
-pp, –preprocess, /PREPROCESS "Fichier" : Prétraite le code source et écrit la sortie dans le "Fichier" spécifié.
Le fichier traité est un fichier unique avec toutes les macro déployées, les directives de compilation sont prises en compte et le multiligne intégré. Cela permet une analyse plus facile d'un fichier source PureBasic, car tout est déployé et disponible dans un format de fichier plat 'flat'. Les commentaires ne sont pas inclus par défaut, sauf avec l'option /COMMENTED. Le fichier prétraité peut être recompilé comme n'importe quel autre fichier source PureBasic pour créer l'exécutable final.
-g, –language, /LANGUAGE \"language\" : utilise la langue spécifiée pour les messages d'erreur du compilateur.
-v, –version, /VERSION : Affiche la version du compilateur.
Exemples :

```
CLI> pbcompiler codesource.pb
```

Le compilateur compile le code 'codesource.pb' et l'exécute.

```
CLI> pbcompiler codesource.pb /DEBUGGER
```

Le compilateur compile le code 'codesource.pb' et l'exécute avec le débogueur actif.

Paramètres du compilateur spécifiques à Windows

/ICON "Icone.ico" : ajoute l'icône spécifiée dans l'exécutable.
/CONSOLE : le fichier de sortie est au format Console. Le format par défaut est Win32.
/DLL : le fichier de sortie est une DLL .
/XP : ajoute le support des thèmes ('skins') pour Windows XP.
/REASM : réassemble le fichier PureBasic.asm file en fichier exécutable. Ceci permet d'utiliser la fonction /COMMENTED, de modifier le fichier de sortie asm et de recréer un exécutable.
/MMX, /3DNOW, /SSE or /SSE2 : génère un exécutable spécifique à un type de processeur. Si une routine est disponible dans une version optimisée pour le type de processeur choisi, elle sera utilisée. Du coup, l'exécutable ne fonctionnera correctement que sur ce type de processeur.
/DYNAMICCPU : génère un exécutable qui contient toutes les versions des routines spécialement optimisées pour un type de processeur. Quand le programme se lance, il déterminera dynamiquement le type du processeur et choisira les routines les plus adaptées. Cela produit un exécutable plus gros mais plus rapide.
/RESOURCE "Fichier" : ajoute un fichier de resource Windows (.rc) à l'exécutable ou la DLL. Ce fichier ne doit pas être un fichier resource compilé, mais un fichier texte contenant des directives. Il est possible de ne specifier qu'un seul fichier resource, mais il peut en inclure d'autre si besoin est, à l'aide des directives adéquates.
/LINKER "FichierCommandes" : spécifie un fichier qui contient des commandes à passer directement au linker. Sur Windows, PureBasic utilise le linker de PellesC (polink), plus d'informations à propos des options possible sont disponible dans l'aide de PellesC.
Les deux options de compilation suivantes sont nécessaires pour créer des programmes fonctionnant sur Microsoft Vista ou au-dessus (Windows 7/8/10). Ces deux options sont en rapport avec le 'manifeste' inclus, donc elles sont ignorées avec les versions plus anciennes de Windows. Si aucune de ces deux options ne s ont utilisées, l'exe s'exécutera en tant qu'utilisateur normal, mais avec la virtualisation activée (redirection du registre et des fichiers). Il est recommandé d'utiliser le commutateur /USER pour désactiver la virtualisation pour tous les programmes qui ont les privilèges d'utilisateur standard, car c'est seulement destinés à la compatibilité des programmes plus anciens. Ces options sont également disponibles dans le menu Compilateur/Options du compilateur de l'IDE.
/ADMINISTRATOR : oblige le programme à demander des privilèges d'administrateur au démarrage. Le programme ne fonctionnera pas sans ça. Cette option est nécessaire. Note : Vous pouvez également

déboguer les programmes avec cette option, mais seulement avec la version autonome du débogueur (car il doit fonctionner avec des droits élevés lui aussi).
/USER : le programme sera exécuté avec les droit de l'utilisateur qui l'a lancé. Virtualisation pour l'exe est désactivée.
Exemples :

```
 CLI> pbcompiler "C:\Project\Source\DLLSource.pb" /EXE
"C:\Projet\projet.dll" /DLL
```

Le compilateur compile le code 'DLLSource.pb' (ici avec le chemin complet) et crée la DLL "projet.dll" à l'emplacement indiqué.

Paramètres du compilateur spécifiques à Linux

-so ou –sharedobject "fichier" : Créer une bibliothèque dynamique (objet partagé).
-h ou –help : affiche une aide simplifiée sur le compilateur.
-mmx, -3dnow, -sse ou -sse2 : génère un exécutable spécifique à un type de processeur. Si une routine est disponible dans une version optimisée pour le type de processeur choisi, elle sera utilisée. Du coup, l'exécutable ne fonctionnera correctement que sur ce type de processeur.
-dc ou –dynamiccpu : génère un exécutable qui contient toutes les versions des routines spécialement optimisées pour un type de processeur. Quand le programme se lance, il déterminera dynamiquement le type du processeur et choisira les routines les plus adaptées. Cela produit un exécutable plus gros mais plus rapide.

Paramètres du compilateur spécifiques à OSX

-dl ou –dylib "Fichier" : crée une bibliothèque dynamique (dylib).
-n ou –icon "Icone.icns" : ajoute une icon à l'application.
-f ou –front : met le processus à l'avant plan lors de son lancement.
-ibp ou –ignorebundlepath : ne pas utiliser le chemin du bundle comme répertoire courant.

Chapitre 23

Les directives du compilateur

Syntax

```
CompilerIf  <expression constante>
  ...
[CompilerElseIf]
  ...
[CompilerElse]
  ...
CompilerEndIf
```

Description

Si <expression constante> est vrai alors le code inclus dans la structure 'If' sera compilé, sinon il sera totalement ignoré.
Cette directive permet de construire des programmes multi-plateformes en personnalisant des parties du code source en fonctions de chaque système d'exploitation.

Exemple

```
1  CompilerIf #PB_Compiler_OS = #PB_OS_Linux
2    ; code spécifique Linux..
3  CompilerElseIf #PB_Compiler_OS = #PB_OS_Windows
4    ; code spécifique pour Windows
5  CompilerEndIf
```

Syntax

```
CompilerSelect <constante numérique>
  CompilerCase <constante numérique>
    ...
  [CompilerElse]
    ...
  [CompilerDefault]
    ...
CompilerEndSelect
```

Description

Fonctionne comme un Select : EndSelect classique en opérant une compilation conditionnelle.
Cette directive permet de construire des programmes multi-plateformes en personnalisant des parties du code source en fonctions de chaque système d'exploitation.

```
1  CompilerSelect #PB_Compiler_OS
2    CompilerCase #PB_OS_MacOS
3      ; du code spécifique à Mac OS X
4    CompilerCase #PB_OS_Linux
5      ; du code spécifique à Linux
6  CompilerEndSelect
```

Syntax

```
CompilerError <message>
CompilerWarning <message>
```

Description

Génère une erreur ou un avertissement du compilateur, comme si c'était une erreur de syntaxe et affiche le message spécifié. Ceci peut être utile pour faire des routines spécifiques et indiquer qu'elles ne sont pas disponibles pour un OS particulier.

Exemple : Génère une erreur

```
1  CompilerIf #PB_Compiler_OS = #PB_OS_Linux
2    CompilerError "Linux n'est pas supporté, désolé."
3  CompilerElse
4    CompilerError "OS Supporté, vous pouvez me mettre en commentaire."
5  CompilerEndIf
```

Exemple : Génère un avertissement

```
1  CompilerIf #PB_Compiler_OS = #PB_OS_Linux
2    CompilerWarning "Linux n'est pas supporté, désolé."
3  CompilerElse
4    CompilerWarning "OS Supporté, vous pouvez me mettre en commentaire."
5  CompilerEndIf
```

Syntax

```
EnableExplicit
DisableExplicit
```

Description

Active ou désactive le mode explicite. Quand il est actif, toutes les variables qui ne sont pas explicitement déclarées avec Define , Global , Protected ou Static ne seront pas acceptées par le compilateur et génèreront une erreur. Cela peut aider à éviter les erreurs de frappe.

Exemple

```
EnableExplicit

Define a

a = 20 ; Ok, car déclaré avec 'Define'
b = 10 ; Erreur
```

Syntax

```
EnableASM
DisableASM
```

Description

Active ou désactive l'assembleur en ligne. Quand il est actif, toutes les commandes assembleur sont incluses directement dans le code source, pour plus d'informations référez vous à la page L'assembleur en ligne .

Exemple

```
; Exemple assembleur x86
;
Test = 10

EnableASM
  MOV dword [v_Test],20
DisableASM

Debug Test ; Affichera 20
```

Constantes prédéfinies

Le compilateur PureBasic déclare automatiquement plusieurs constantes avant chaque compilation pour donner plus d'informations sur l'environnement de développement :

```
  #PB_Compiler_OS: Permet de savoir sur quelle plateforme est exécuté
 le compilateur
  #PB_OS_Windows : Le compilateur est exécuté sur Windows
  #PB_OS_Linux   : Le compilateur est exécuté sur Linux
  #PB_OS_MacOS   : Le compilateur est exécuté sur Mac OS X

#PB_Compiler_Processor : Détermine le type de processeur pour lequel
 le programme est créé. Il peut être l'un des suivants:
  #PB_Processor_x86     : Architecture de processeur x86 (aussi
 appelé IA-32 ou x86-32)
  #PB_Processor_x64     : Architecture de processeur x86-64 (aussi
 appelé x64, AMD64 ou Intel64)

#PB_Compiler_ExecutableFormat : Détermine le format de l'exécutable.
 Il peut être l'un des suivants:
  #PB_Compiler_Executable : Exécutable standard
```

```
    #PB_Compiler_Console    : Exécutable console (Uniquement sur
   Windows, les autres OS utiliseront le format exécutable standard)
    #PB_Compiler_DLL        : DLL (dynlib sur MacOS X et objet partagé
   sur Linux)


  #PB_Compiler_Date     : La date actuelle, au moment de la
    compilation, au format date
 PureBasic.
  #PB_Compiler_File     : Chemin complet et nom du fichier en cours de
    compilation, utile pour debogguer
.
  #PB_Compiler_FilePath : Chemin complet du fichier en cours de
    compilation, utile pour debogguer
.
  #PB_Compiler_Line     : Numéro de la ligne du fichier en cours de
    compilation, utile pour debogguer
.
  #PB_Compiler_Procedure: Nom de la procédure actuelle, si la ligne est
    à l'intérieur d'une procédure
.
  #PB_Compiler_Module   : Nom du module courant, si la ligne est à
    l'intérieur d'un module
.
  #PB_Compiler_Version  : Version du compilateur, nombre entier sous la
    forme '420' pour la version 4.20.
  #PB_Compiler_Home     : Chemin complet du répertoire PureBasic, utile
    pour localiser des fichiers inclus

  #PB_Compiler_Debugger : Egal à 1 si le débogueur
 est activé, égal 0 sinon. Quand un exécutable est
                          créé, le débogueur est toujours désactivé
    (cette constante sera égale à 0).
  #PB_Compiler_Thread   : Egal à 1 si l'exécutable est compilé en mode
    'multi-threadé', égal à 0 sinon.
  #PB_Compiler_Unicode  : Egal à 1 si l'exécutable est compilé en mode
    unicode
, égal à 0 sinon.
  #PB_Compiler_LineNumbering : Egal à 1 si l'exécutable est compilé
    avec l'option 'Activer le numéro de ligne pour OnError
', égal à 0 sinon.
  #PB_Compiler_InlineAssembly: Egal à 1 si l'exécutable est compilé
    avec l'option 'Activer l'assembleur en ligne
', égal à 0 sinon.
  #PB_Compiler_EnableExplicit: Egal à 1 si l'exécutable est compilé en
    mode 'explicit', égal à 0 sinon.
  #PB_Compiler_IsMainFile    : Egal à 1 si l'exécutable en cours de
    compilation est le fichier principal, égal à 0 sinon.
  #PB_Compiler_IsIncludeFile : Egal à 1 si l'exécutable en cours de
    compilation a été inclus par un autre fichier, égal à 0 sinon.
```

Chapitre 24

Les fonctions du compilateur

Syntax

```
Bool(<expression booléenne>)
```

Description

Bool permet d'évaluer une expression booléenne en dehors des opérateurs conditionnels réguliers comme If, While, Until, etc.

Arguments

expression booléenne L'expression booléenne à tester.

Valeur de retour

Renvoie #True si l'expression booléenne est vraie, #False sinon.

Exemple

```
1  Salut$ = "Salut"
2  LeMonde$ = "Le Monde"
3
4  Debug Bool(Salut$ = "Salut")  ; Affichera 1
5  Debug Bool(Salut$ <> "Salut" Or LeMonde$ = "Le Monde") ; Affichera 1
```

Exemple

```
1  Procedure Chiffre(char.c)
2    ProcedureReturn Bool(char >= '0' And char <= '9')
3  EndProcedure
4
5  Debug Chiffre('0')
6  Debug Chiffre('1')
7  Debug Chiffre('a')
8  Debug Chiffre('z')
```

Syntax

```
Resultat = Defined(Nom, Type)
```

Description

Defined détermine si un objet tel qu'une structure , interface , variables etc. est déjà défini dans le programme.

Arguments

Nom Le nom de l'objet.
Le paramètre 'Nom' doit être spécifié sans aucune forme de décoration (sans le '#' pour une constante , sans les '()' pour un tableau , une liste , une map ou une procédure).

Type Peut prendre une des valeurs suivantes :

```
#PB_Constant
#PB_Variable
#PB_Array
#PB_List
#PB_Map
#PB_Structure
#PB_Interface
#PB_Procedure
#PB_Function
#PB_OSFunction
#PB_Label
#PB_Prototype
#PB_Module
#PB_Enumeration
```

Valeur de retour

Aucune.

Exemple

```
1   #PureConstante = 10
2
3   CompilerIf Defined(PureConstante, #PB_Constant)
4     Debug "La constante 'PureConstante' est déjà déclarée"
5   CompilerEndIf
6
7   Test = 25
8
9   CompilerIf Defined(Test, #PB_Variable)
10    Debug "La variable 'Test' est déjà déclarée"
11  CompilerEndIf
```

Syntax

```
Resultat = Subsystem(<expression texte constante>)
```

Description

Subsystem permet de savoir si un sous-système est utilisé pour le programme en cours de compilation.

Arguments

expression texte constante Le nom de sous-système.
Peut prendre une des valeurs suivantes : Sensible à la casse.

```
OpenGL
DirectX11
```

Valeur de retour

Renvoie une valeur ni nulle si le sous système est utilisé, zéro sinon.

Exemple

```
1  CompilerIf Subsystem("OpenGL")
2    Debug "Compilation avec le sous-système OpenGL"
3  CompilerEndIf
```

Syntax

```
Resultat = OffsetOf(Structure\Champ)
Resultat = OffsetOf(Interface\Fonction())
```

Description

La commande OffsetOf permet de déterminer la position en mémoire d'un champ d'une structure ou la position en mémoire d'une fonction dans le cas d'une interface (soit IndexDeLaFunction*SizeOf(Integer)).

Arguments

Structure\Champ ou Interface\Fonction() Le champ de la structure ou la fonction de l'interface.

Valeur de retour

Renvoie l'index du champ ou de la fonction, zéro sinon.

Exemple

```
1  Structure Personne
2    Nom.s
3    Prenom.s
4    Age.w
5  EndStructure
```

```
6
7   Debug OffsetOf(Personne\Age) ; Affichera 8 car une 'string' occupe 4
     octets en mémoire
8                                ;(16 avec un compilateur 64 bits, car
     une 'string' occupe 8 octets en mémoire)
9
10  Interface ITest
11    Creer()
12    Detruire(Options)
13  EndInterface
14
15  Debug OffsetOf(ITest\Detruire()) ; Affichera 4
```

Syntax

```
Resultat = SizeOf(Type)
```

Description

La commande SizeOf permet de renvoyer la taille en octets que prendra une structure (ne fonctionne pas avec les types de base tels que les 'word', les 'float', etc.), une interface ou même une variable.
Comme SizeOf est une fonction de compilation, elle ne fonctionne pas avec les tableaux, les Listes ou les Maps. Utilisez ArraySize() , ListSize() ou MapSize() à la place.

Arguments

Type Le type de l'objet.

Valeur de retour

La taille de l'objet en mémoire, en octets

Remarques

C'est très utile dans de nombreux cas, notamment lors de l'utilisation des commandes API.
Note : Une variable de type caractère (CHARACTER) (.c) est unicode et occupe 2 octets et une variable de type ASCII (.a) n'occupe qu'1 octet.
Note : Si une variable et une structure ont le même nom, la structure aura la priorité sur la variable.

Exemple : 1

```
1   VariableCaractere.c = '!'
2   Debug SizeOf(VariableCaractere) ; affiche 2, c'est à dire 2 octets
3   Debug SizeOf(CHARACTER)         ; affiche 2, c'est à dire 2 octets
4
5   VariableAscii.a = '!'
6   Debug SizeOf(VariableAscii)     ; affiche 1, c'est à dire 1 octets
7   Debug SizeOf(ASCII)             ; affiche 1, c'est à dire 1 octets
```

Exemple : 2

```
1   Structure Personne
2     Nom.s
3     Prenom.s
4     Age.w
5   EndStructure
6
7   Debug "La taille d'une personne est "+Str(Sizeof(Personne))+" octets"
      ; Affichera 10 (4+4+2)
8
9   Jean.Personne\Nom = "Jean"
10
11  Debug SizeOf(Jean) ; Affichera 10 aussi
```

Syntax

```
Resultat = TypeOf(Objet)
```

Description

TypeOf permet de déterminer le type d'une variable , ou d'un champ de structure .

Arguments

Objet L'objet à utiliser.

Valeur de retour

Le type de l'objet.
Le type peut être une des valeurs suivantes :

```
#PB_Byte
#PB_Word
#PB_Long
#PB_String
#PB_Structure
#PB_Float
#PB_Character
#PB_Double
#PB_Quad
#PB_List
#PB_Array
#PB_Integer
#PB_Map
#PB_Ascii
#PB_Unicode
#PB_Interface
```

Exemple

```
Structure Personne
  Nom.s
  Prenom.s
  Age.w
EndStructure

If TypeOf(Personne\Age) = #PB_Word
  Debug "Age est un 'Word'"
EndIf

Surface.f
If TypeOf(Surface) = #PB_Float
  Debug "Surface est un 'Float'"
EndIf
```

Syntax

```
InitializeStructure(*Memoire, Structure)
```

Description

Initialise un objet structuré en mémoire.
En fait, cette fonction initialise les membres d'une structure . Ces membres sont de type Array , List ou Map mais les autres types ne sont pas affectés (.s .l, .i, etc.).

Arguments

***Memoire** L'adresse mémoire à utiliser.

Structure Le nom de la structure qui doit être utilisé pour effectuer l'initialisation.
Il n'y a pas de contrôle pour s'assurer que la zone mémoire corresponde à la structure.

Valeur de retour

Aucune.

Remarques

Attention : plusieurs appels à InitializeStructure crée une fuite de mémoire parce que les anciens membres de la structure ne sont pas libérés de la mémoire. ClearStructure doit être appelée avant d'appeler une nouvelle fois InitializeStructure.
Cette fonction est pour les utilisateurs avancés et doit être utilisée avec précaution.
Pour allouer une structure dynamique, utiliser AllocateStructure() ().

Exemple

```
Structure Personne
  Prenom$
  Age.l
  List Amis.s()
EndStructure
```

```
6
7   *Etudiant.Personne = AllocateMemory(SizeOf(Personne))
8   InitializeStructure(*Etudiant, Personne)
9
10  ; Maintenant, la liste est prête à l'emploi
11  ;
12  AddElement(*Etudiant\Amis())
13  *Etudiant\Amis() = "John"
14
15  AddElement(*Etudiant\Amis())
16  *Etudiant\Amis() = "Yann"
17
18  ; Affichage du contenu de la liste
19  ;
20  ForEach *Etudiant\Amis()
21    Debug *Etudiant\Amis()
22  Next
```

Syntax

```
CopyStructure(*Source, *Destination, Structure)
```

Description

Copie une structure en mémoire vers une autre.

Arguments

***Source** L'adresse mémoire contenant la structure à copier.

***Destination** L'adresse mémoire de la copie.

Structure Le nom de la structure qui doit être utilisé pour effectuer la copie.

Valeur de retour

Aucune.

Remarques

C'est particulièrement utile lors de l'utilisation de mémoire dynamique avec les pointeurs. Chaque champ de la structure sera dupliqué, y compris les tableaux dynamiques, les listes ou les maps . La structure de destination sera automatiquement effacée avant de faire la copie, il n'est pas nécessaire d'appeler ClearStructure avant CopyStructure.
Attention : La destination doit être une zone de mémoire de structure valide, ou une zone mémoire effacée. Si la zone de mémoire n'est pas effacée, cela pourrait provoquer un crash, car des valeurs aléatoires seront utilisées.
Il n'y a pas de contrôle pour s'assurer que les deux zones mémoires sont bien du type 'Structure', donc il est impératif de manipuler cette commande avec précaution.

Exemple

```
Structure Personne
  Prenom$
  Nom$
  Map Amis$()
  Age.l
EndStructure

Etudiant.Personne\Prenom$ = "Paul"
Etudiant\Nom$ = "Morito"
Etudiant\Amis$("Tom") = "Jones"
Etudiant\Amis$("Jim") = "Doe"

CopyStructure(@Etudiant, @EtudiantCopy.Personne, Personne)

Debug EtudiantCopy\Prenom$
Debug EtudiantCopy\Nom$
Debug EtudiantCopy\Amis$("Tom")
Debug EtudiantCopy\Amis$("Jim")
```

Syntax

```
Resultat = ClearStructure(*Memoire, Structure)
```

Description

Vide la zone mémoire structurée et met la valeur de tous les champs à zéro.

Arguments

***Memoire** L'adresse mémoire contenant la structure à effacer.

Structure Le nom de la structure qui sera utilisée pour effectuer le nettoyage.

Valeur de retour

Aucune.

Remarques

C'est particulièrement utile quand la structure contient des chaînes de caractères, des tableaux , des listes ou des maps qui ont été alloués en interne par PureBasic. Tous les champs seront mis à zéro, même les types natifs comme long, integer, etc.
Il n'y a pas de contrôle pour s'assurer que la zone mémoire est bien du type 'Structure' spécifié, donc il est impératif de manipuler cette commande avec précaution. Cette fonction est réservée aux utilisateurs avancés.

Exemple

```
Structure Personne
  Prenom$
  Nom$
```

```
    Age.l
  EndStructure

  Etudiant.Personne\Prenom$ = "Paul"
  Etudiant\Nom$ = "Morito"
  Etudiant\Age = 10

  ClearStructure(@Etudiant, Personne)

  ; Affichera des chaines vide, car la structure entiere a ete videe.
   Tous les autres champs ont ete remis a zero
  ;
  Debug Etudiant\Prenom$
  Debug Etudiant\Nom$
  Debug Etudiant\Age
```

Syntax

```
ResetStructure(*Memoire, Structure)
```

Description

ResetStructure, vide la zone mémoire structurée et l'initialise pour être prête à être utilisée.

Arguments

***Memoire** L'adresse mémoire contenant la structure à réinitialiser.

Structure Le nom de la structure utilisée.

Valeur de retour

Aucune.

Remarques

C'est particulièrement utile quand la structure contient des chaînes de caractères, des tableaux , des listes ou des maps qui ont été alloués en interne par PureBasic. Tous les champs seront mis à zéro, même les types natifs comme long, integer, etc.
Il n'y a pas de contrôle pour s'assurer que la zone mémoire est bien du type 'Structure' spécifié, donc il est impératif de manipuler cette commande avec précaution. Cette fonction est réservée aux utilisateurs avancés.

Exemple

```
  Structure Personne
    Map Amis.s()
  EndStructure

  Henri.Personne\Amis("1") = "Paul"

  ResetStructure(@Henri, Personne)
```

```
8
9   ; Affiche une chaîne vide car l'ensemble de la structure a été
      réinitialisée.
10  ; La map est encore utilisable, mais vide.
11  ;
12  Debug Henri\Amis("1")
```

Chapitre 25

Data

Introduction

PureBasic autorise l'utilisation de données pour stocker des blocs d'informations prédéfinies dans votre programme. Ceci est utile pour disposer de valeurs par défaut (messages textuels par exemple) ou dans un jeu pour définir le cheminement prédéfini d'un sprite.
DataSection doit être placé en tête du bloc de données. Tous les labels et composants data sont stockés dans cette partie data dont l'accès est plus rapide que la zone de code. Data est utilisé pour entrer des données. EndDataSection doit être spécifié si du code à exécuter est situé après. Il est intéressant de pouvoir placer plusieurs zones de données à différents endroits du code source. Les données peuvent être chargées à l'aide des commandes Restore et Read.

Syntax

```
DataSection
```

Description

Début d'une zone de données.

Exemple

```
1  DataSection
2    DonneesNumeriques:
3      Data.l 100, 200, -250, -452, 145
4
5    DonneesTexte:
6      Data.s "Bonjour", "Qu'est-ce", "que ", "c'est ?"
7  EndDataSection
```

Syntax

```
EndDataSection
```

Description

Fin d'une zone de données.

Exemple

```
DataSection
  DonneesNumeriques:
    Data.l 100, 200, -250, -452, 145

  DonneesTexte:
    Data.s "Bonjour", "Qu'est-ce", "que ", "c'est ?"
EndDataSection
```

Syntax

```
Data.NomType
```

Description

Définit les données. Le type peut être choisi parmi les types natifs (integer, long, word, byte, ascii, unicode, float, double, quad, character, string). Un nombre quelconque de données peut être placé sur une même ligne, chacune étant séparée par une virgule.

Exemple

```
Data.l 100, 200, -250, -452, 145
Data.s "Bonjour", "Qu'est", "ce que ", "c'est ?"
```

Pour les programmeurs chevronnés : il est aussi possible de mettre l'adresse d'une procédure ou d'un label avec Data si le type 'entier' (integer .i) est utilisé. Sur un système 32 bits les adresses seront stockées sur 4 octets et sur 8 octets sur un système 64 bits.
Par exemple, des tables de fonctions virtuelles peuvent être créées facilement.

Exemple

```
Procedure Max(Nombre, Nombre2)
EndProcedure

Etiquette:

DataSection
  Data.i ?Etiquette, @Max()
EndDataSection
```

Exemple

```
Interface MonObjet
   FaireCeci()
   FaireCela()
EndInterface

Procedure Ceci(*Self)
   MessageRequester("MonObjet", "Ceci")
EndProcedure
```

```
9
10  Procedure Cela(*Self)
11     MessageRequester("MonObjet", "Cela")
12  EndProcedure
13
14  m.MonObjet = ?VTable
15
16  m\FaireCeci()
17  m\FaireCela()
18
19
20  DataSection
21     VTable:
22        Data.i ?Procedures
23     Procedures:
24        Data.i @Ceci(), @Cela()
25  EndDataSection
```

Syntax

```
Restore label
```

Description

Ce mot clef permet de placer un indicateur de début de zone pour la commande Read. Le label utilisé par cette commande doit être déclaré dans le bloc DataSection, car il sera déplacé lors de la création de l'exécutable.

Exemple

```
1   Restore DonneesTexte
2   Read.s MonPremierTexte$
3   Read.s MonSecondTexte$
4
5   Restore DonneesNumeriques
6   Read.l a
7   Read.l b
8
9   Debug MonPremierTexte$
10  Debug a
11
12  End
13
14  DataSection
15
16    DonneesNumeriques:
17      Data.l 100, 200, -250, -452, 145
18
19    DonneesTexte:
20      Data.s "Bonjour", "Qu'est-ce", "que ", "c'est ?"
21  EndDataSection
```

Syntax

```
Read[.<type>] <variable>
```

Description

Lit la donnée suivante. Le pointeur peut être modifié en utilisant la commande Restore. Par défaut, la donnée suivante est la première donnée déclarée.

Exemple

```
1  Restore DonneesTexte
2  Read.s MonPremierTexte$
3  Read.s MonSecondTexte$
4
5  Restore DonneesNumeriques
6  Read.l a
7  Read.l b
8
9  Debug MonPremierTexte$
10 Debug MonSecondTexte$
11 Debug a
12 Debug b
13
14 End
15
16 DataSection
17
18   DonneesNumeriques:
19     Data.l 100, 200, -250, -452, 145
20
21   DonneesTexte:
22     Data.s "Bonjour", "Qu'est-ce", "que ", "c'est ?"
23 EndDataSection
```

Chapitre 26

Commandes de débogage

Introduction

La description complète des fonctionnalités du débogueur se trouve dans les chapitres Utilisation du débogueur et Utiliser les outils de déboguage .
Une bibliothèque Debugger est également disponible pour controler le comportement du débogueur à partir du code source.

Syntax

```
CallDebugger
```

Description

Appel du ”débogueur” et arrêt immédiat du programme au point courant du code.

Syntax

```
Debug <expression> [, NiveauDebug]
```

Description

Affiche la fenêtre DebugOutput et le résultat correspondant. L'expression peut être toute expression valide en PureBasic, de forme numérique ou chaine. Un point important est que toute commande Debug et les expressions associées sont totalement ignorées (non compilées) si le débogueur est désactivé. Cela singifie qu'il n'est pas nécessaire de passer les instruction Debug en commentaires lors de la création de l'exécutable final tout en ayant la possibilité de tracer facilement l'exécution du programme pour le développeur.
'NiveauDebug' est le niveau de priorité des messages du débogueur. Tous les messages (avec un niveau non spécifié) sont affichés automatiquement. Lorsqu'un niveau est spécifié alors le message correspondant ne sera affiché que si le niveau de debug courant est égal ou supérieur au niveau associé au message. Cela permet de réaliser une traçabilité hiérarchisée en affichant des informations de plus en plus précises en fonction de la valeur 'NiveauDebug' utilisée.

Syntax

```
DebugLevel <expression constante >
```

Description

Fixe le niveau courant pour les messages 'Debug'.
Note : Le niveau est fixé au moment de la compilation, ce qui signifie que vous devez mettre la commande DebugLevel avant les commandes debug pour être sûr qu'elles seront bien toutes affectées.

Syntax

```
DisableDebugger
```

Description

Interromps l'utilisation du débogueur sur les lignes du code qui suivent cette commande.

Syntax

```
EnableDebugger
```

Description

Active l'utilisation du débogueur sur les lignes de code qui suivent cette commande (lorsque le débogueur a été préalablement interrompu par la commande DisableDebugger).

Chapitre 27

Define

Syntax

```
Define.<type> [<variable> [= <expression>], <variable> [=
    <expression>], ...]
```

Description

Si aucune <variable> n'est spécifiée, Define est utilisé pour changer le type par défaut des variables qui seront ensuite utilisées sans déclaration (y compris les paramètres des procedures et les paramètres des méthodes dans les interfaces ainsi que les données lues avec le mot clé Read .
Le type par défaut initial est le type integer (.i). Chaque variable peut avoir une valeur assignée par défaut.
Define peut également être utilisé avec les tableaux , les listes et les maps .

Exemple

```
1  d = e + f
2  Define.w
3  a = b + c
```

Les variables d, e et f sont créées avec le type integer, puisqu'il n'y a pas eu de type spécifié. Les variables a, b et c sont des mots signés (.w) car aucun type ne leur est spécifié, le type par défaut ayant été passé à .w.
Si des variables sont précisées avec la commande Define le type par défaut n'est pas changé seules les variables nommées utilisant le type indiqué.

Exemple

```
1  Define.b a, b = 10, c = b*2, d
```

a,b,c,d sont de type octet (.b)

Syntax

```
Define <variable>.<type> [= <expression>] [, <variable>.<type> [=
    <expression>], ...]
```

Description

Autre possibilité pour la déclaration des variables avec Define.

Exemple

```
1  Define MonChar.c ; Caractère
2  Define MonLong.l ; Double Mots
3  Define MonWord.w ; Mot
4
5  Debug SizeOf(MonChar)   ; Affichera 2 (à cause du mode unicode)
6  Debug SizeOf(MonLong)   ; Affichera 4
7  Debug SizeOf(MonWord)   ; Affichera 2
```

Chapitre 28

Dim

Syntax

```
Dim nom.<type>(<expression>, [<expression>], ...)
```

Description

Dim est utilisé pour dimensionner un nouveau tableau.
Un tableau peut être composé d'éléments de type basique connu sous PureBasic, incluant les structures et les types définis par l'utilisateur.
Attention : Les éléments d'un tableau doivent tous être de même type.
Une fois le tableau créé il peut être redimensionné mais son contenu sera alors effacé, sauf si ReDim est utilisé.
Les tableaux sont alloués dynamiquement ce qui signifie que le dimensionnement peut se faire à partir d'une variable ou d'une expression.
Les tableaux sont toujours locaux par défaut, donc pour accéder à partir d'une procedure à un tableau défini dans le code source principal du programme, l'utilisation de Global ou Shared est requise. Il est également possible de passer un tableau en paramètre d'une procédure à l'aide du mot clef Array. Il sera passé "par référence" (ce qui signifie que le tableau ne sera pas copié, et les fonctions dans la procédure manipulerons le tableau original).
Pour consulter toutes les commandes relatives aux tableaux comme la copie, la destruction ou l'obtention de la taille d'un tableau, voir la bibliothèque Array .
Pour effacer le contenu complet d'un tableau et libérer la mémoire qu'il occupe, utilisez FreeArray() .
Si Dim est utilisé sur un tableau existant, il réinitialise son contenu à zéro.
Utilisez la commande Swap pour permuter le contenu de tableaux rapidement.
Note : La vérification des accès à un tableau est effectuée uniquement quand le débogueur est activé.

Exemple

```
1  Dim MonTableau(41)
2  MonTableau(0) = 1
3  MonTableau(1) = 2
```

Exemple : Tableau à dimensions multiples

```
1  Dim TableauMultiple.b(NbColonnes, NbLignes)
2  TableauMultiple(10, 20) = 10
3  TableauMultiple(20, 30) = 20
```

Exemple : Tableau en paramètre d'une procédure

```
1   Procedure fill(Array Nombres(1), Longueur)  ; Le 1 représente le
     nombre de dimensions du tableau
2     For i = 0 To Longueur
3       Nombres(i) = i
4     Next i
5   EndProcedure
6
7   Dim Nombres(10)
8   fill(Nombres(), 10)  ; Le tableau Nombres() est passé en paramètre
9
10  Debug Nombres(5)
11  Debug Nombres(10)
```

Syntax

```
ReDim nom.<type>(<expression>, [<expression>], ...)
```

Description

ReDim permet de redimensionner un tableau déjà déclaré tout en préservant son contenu. La nouvelle taille peut être plus grande ou plus petite, mais le nombre de dimensions ne peut pas être modifié.
Si ReDim est utilisé sur un tableau à plusieurs dimensions, seule la dernière dimension peut être changée.

Exemple

```
1   Dim MonTableau.l(1) ; nous avons 2 éléments
2   MonTableau(0) = 1
3   MonTableau(1) = 2
4
5   ReDim MonTableau(4) ; Maintenant nous avons 5 éléments
6   MonTableau(2) = 3
7
8   For k = 0 To 2
9     Debug MonTableau(k)
10  Next
```

Chapitre 29

Construire une DLL

Introduction

PureBasic permet de créer des DLL Microsoft Windows (DLL : Dynamic Linked Library), des objets partagés (.so) sous Linux, et des bibliothèques dynamiques (.dylib) sous MacOS X. Le code d'une DLL est de même nature que le code PureBasic excepté qu'aucun code ne peut exister en dehors d'une procédure. Tout le code est intégré dans des procédures, sauf la déclaration des variables globales et des structures.
L'avantage d'une DLL est de pouvoir partager du code déjà compilé avec un programme tiers. Pour cela il faut déclarer la DLL avec le mot clef ProcedureDLL au lieu de Procedure . Si la procédure partagée doit être au format 'CDecl' (ce qui n'est pas le cas pour les DLL standards de Windows), le mot clef ProcedureCDLL doit être utilisé.
De même Declare doit être remplacé par DeclareDLL ou DeclareCDLL.
Avant de compiler a DLL, il est necessaire de sélectionner 'Shared DLL' comme format de sortie dans le menu 'Compiler\Option' de l'éditeur PureBasic (ou d'utiliser le commutateur /DLL dans la ligne de commande). Une fois compilé, une DLL apparait avec le même nom que le fichier PureBasic.

Exemple

```
1   ; Créer et enregistrer ce fichier sous le nom 'PureBasic.pb'
2   ; La DLL 'PureBasic.dll' sera créée.
3
4   ProcedureDLL MaFonction()
5     MessageRequester("Bonjour", "Voici une DLL PureBasic !", 0)
6   EndProcedure
7
8   ; Créer un deuxième fichier PureBasic avec le code suivant:
9   ; Voici le programme client qui utilise la DLL
10  ; Compiler et executer ce programme dans le dossier
11  ; contenant la dll 'PureBasic.dll'.
12
13  If OpenLibrary(0, "PureBasic.dll")
14    CallFunction(0, "MaFonction")
15    CloseLibrary(0)
16  EndIf
```

Pour les programmeurs avancés, il existe 4 procédures spéciales qui sont appelées automatiquement par l'OS lorsque l'un des évènements suivants surviennent :
- une DLL est attachée à un nouveau process
- une DLL est détachée d'un process
- une DLL est attachée à un nouveau thread

- une DLL est détachée d'un thread
Pour gérer cela, il est possible de déclarer 4 procédures spéciales nommées : AttachProcess(Instance), DetachProcess(Instance), AttachThread(Instance) et DetachThread(Instance). Cela signifie que ces 4 noms sont réservés et ne peuvent être utilisés par le programmeur pour d'autres usages.
Notes à propos de la création des DLL's :
- La déclaration des tableaux avec Dim , des listes avec NewList ou des maps avec NewMap doit toujours être faite dans la procédure 'AttachProcess()'.
- Ne pas écrire de code en dehors des procédures. La seule exception est la déclaration des variables et des structures.
- Les routines d'initialisation DirectX ne doivent pas être dans la procédure 'AttachProcess()'.
A propos du renvoi de chaîne de caractères (string) par une DLL :
Pour qu'une fonction puisse renvoyer une string, elle doit être déclarée en global .

Exemple

```
1  Global ReturnString$
2
3  ProcedureDLL.s MaFonction(var.s)
4    ReturnString$ = var + " test"
5    ProcedureReturn ReturnString$
6  EndProcedure
```

Si la chaîne de caractères n'est pas déclarée en global alors elle sera locale à la procédure et donc libérée à la fin de la procédure. Elle ne pourra pas être utilisée par le programme appelant la fonction.
Quand CallFunction() ou une fonction similaire de type CallXXX est utilisée dans un programme pour appeler une fonction dans une DLL, le programme reçoit un pointeur vers une chaîne de caractères, qui pourra être lu avec PeekS() .

Exemple

```
1  String.s = PeekS(CallFunction(0, "NomFonction", Parametre1,
     Parametre2))
```

Vous trouverez ci dessous un exemple complet :

Chapitre 30

Enumérations

Syntax

```
Enumeration [nom] [<constante> [Step <constante>]]
  #Constante1
  #Constante2 [= <constante>]
  #Constante3
  ...
EndEnumeration
```

```
EnumerationBinary [nom] [<constante>]
  #Constante1
  #Constante2 [= <constante>]
  #Constante3
  ...
EndEnumeration
```

Description

Les énumérations sont très pratiques pour déclarer rapidement une série de constantes sans s'occuper de leur valeur numérique. La première constante de l'énumération prendra la valeur 0, la constante suivante prendra la valeur 1 etc. Il est possible de changer la valeur de départ de l'énumération et d'ajuster la valeur utilisée pour l'incrémentation de chaque constante. Si nécessaire, il est possible d'affecter directement une valeur numérique à une constante (grâce à l'opérateur '=') et les constantes suivantes utiliseront cette nouvelle valeur comme valeur de base. Comme les énumerations n'acceptent que les nombres entiers, les nombres flottants seront arroundis à l'entier le plus proche.
Un 'nom' peut être configuré pour identifier une énumération et permettre de l'interrompre puis de la poursuivre plus tard. C'est utile pour regrouper des objets dans une même énumération tout en les déclarant dans différents endroits du code.
Pour les utilisateurs avancés seulement : La constante réservée #PB_Compiler_EnumerationValue stocke la prochaine valeur qui sera utilisée par l'énumération. Cela peut être utile pour connaître la valeur de la dernière énumération ou pour chaîner plusieurs énumérations.
EnumerationBinary peut être utilisé pour créer des énumérations appropriées pour les options (flags). La première valeur de l'élément est 1.

Exemple : Enumération simple

```
1  Enumeration
2    #GadgetInfo ; égale à 0
```

```
    #GadgetText ; égale à 1
    #GadgetOK   ; égale à 2
  EndEnumeration
```

Exemple : Enumération avec un pas déterminé

```
  Enumeration 20 Step 3
    #GadgetInfo ; égale à 20
    #GadgetText ; égale à 23
    #GadgetOK   ; égale à 26
  EndEnumeration
```

Exemple : Enumération avec un changement dynamique

```
  Enumeration
    #GadgetInfo       ; égale à 0
    #GadgetText = 15 ; égale à 15
    #GadgetOK         ; égale à 16
  EndEnumeration
```

Exemple : Enumérations nommées

```
  Enumeration Gadget
    #GadgetInfo ; égale à 0
    #GadgetText ; égale à 1
    #GadgetOK   ; égale à 2
  EndEnumeration

  Enumeration Window
    #FirstWindow  ; égale à 0
    #SecondWindow ; égale à 1
  EndEnumeration

  Enumeration Gadget
    #GadgetCancel ; égale à 3
    #GadgetImage  ; égale à 4
    #GadgetSound  ; égale à 5
  EndEnumeration
```

Exemple : Obtenir la prochaine valeur d'énumération

```
  Enumeration
    #GadgetInfo ; égale à 0
    #GadgetText ; égale à 1
    #GadgetOK   ; égale à 2
  EndEnumeration

  Debug "La prochaine valeur d'énumération est : " +
   #PB_Compiler_EnumerationValue ; affiche 3
```

Exemple : Enumération binaire

```
1  EnumerationBinary
2    #Flags1 ; égale à 1
3    #Flags2 ; égale à 2
4    #Flags3 ; égale à 4
5    #Flags4 ; égale à 8
6    #Flags5 ; égale à 16
7  EndEnumeration
```

Chapitre 31

For : Next

Syntax

```
For <variable> = <expression1> To <expression2> [Step <constante>]
  ...
Next [<variable>]
```

Description

La fonction For : Next est utilisée pour produire une boucle dans le programme, avec les paramètres définis. A chaque cycle, <variable> est incrémentée de 1 (ou d'une valeur correspondant au pas indiqué dans Step <constante>). Le première valeur de <variable> est <expression1>. La boucle est interrompue dès que la valeur de <variable> est supérieure la valeur de <expression2>.
La commande Break permet de quitter à n'importe quel moment une ou plusieurs boucles. la commande Continue permet de passer directement à la prochaine itération de la boucle.
La boucle For : Next fonctionne uniquement avec des valeurs entières, aussi bien pour les expressions que pour la constante Step. La constante Step peut aussi être négative.

Exemple

```
1  For k = 0 To 10
2    Debug k
3  Next
```

Dans cet exemple, le programme bouclera 11 fois (de 0 à 10) et sortira.

Exemple

```
1  For k = 10 To 1 Step -1
2    Debug k
3  Next
```

Dans cet exemple, le programme bouclera 10 fois (de 10 à 1) et sortira.

Exemple

```
a = 2
b = 3
For k = a+2 To b+7 Step 2
  Debug k
Next k
```

Ici, le programme bouclera 4 fois avant de sortir (k est augmentée de 2 à chaque cycle et prend donc successivement les valeurs 4, 6, 8 et 10=. La lettre k après le mot clef Next indique que ce Next ferme la boucle "For k". Si un autre nom de variable est utilisé, le compilateur génèrera une erreur. Cela est utile pour la lisibilité d'un code comprenant des boucles imbriquées.

Exemple

```
For x=0 To 319
  For y=0 To 199
    Plot(x,y)
  Next y
Next x
```

Note : Gardez à l'esprit que la valeur de <expression2> peut également être changée dans la boucle For : Next, ce qui peut engendrer une boucle sans fin si la valeur n'est pas correcte.

Chapitre 32

ForEach : Next

Syntax

```
ForEach Liste() Ou Map()
  ...
Next [Liste() Ou Map()]
```

Description

ForEach énumère tous les élèments d'une liste ou d'une map . Si la liste ou la map est vide, ForEach : Next quitte immédiatement, sans entrer dans la boucle.
Lors du l'utilisation en conjonction avec une liste : comme la boucle se termine seulement lorsque le dernier élément de la liste est atteint (en terme de position), il est tout à fait possible de supprimer ou d' ajouter des éléments durant un cycle de boucle. De même il est permis de changer l'élément courant avec ChangeCurrentElement() . Après l'un de ces changements, le prochain cycle de boucle continue avec l'élément qui suit l'élément courant.
Il est possible de quitter une boucle ForEach : Next à tout moment à l'aide de la commande Break . La commande Continue permet de passer directement à l'itération suivante, sans exécuter la fin du code contenu dans la boucle.

Exemple : Liste

```
1   NewList Nombre()
2
3   AddElement(Nombre())
4   Nombre() = 10
5
6   AddElement(Nombre())
7   Nombre() = 20
8
9   AddElement(Nombre())
10  Nombre() = 30
11
12  ForEach Nombre()
13    Debug Nombre() ; Affichera 10, 20 et 30
14  Next
```

Exemple : Map

```
NewMap Pays.s()

Pays("US") = "United States"
Pays("FR") = "France"
Pays("DE") = "Allemagne"

ForEach Pays()
  Debug Pays()
Next
```

Chapitre 33

Règles de syntaxe générales

PureBasic a défini des règles qui ne changent jamais.

> Commentaires

Les commentaires sont signalés par ';' . Tout le texte situé après le caractère ';' est ignoré par le compilateur jusqu'à la fin de la ligne.

Exemple

```
1  If a = 10 ; Ceci est un commentaire pour indiquer quelque
     chose.
```

> Mots clé (Keywords)

Tous les **mots clés** sont utilisés pour des choses générales à l'intérieur de PureBasic, comme la création de tableaux (Dim) ou des listes (NewList), ou le contrôle du déroulement du programme (If : Else : EndIf). Ils ne sont pas suivis par les parenthèses '()', qui sont généralement utilisées par PureBasic pour les **fonctions**.

Exemple

```
1     If a = 1       ; If, Else et EndIf sont des mots clés;
    contrairement à 'a'
2       ...          ; qui est une variable utilisée dans une
    expression ('a = 1').
3     Else
4       ...
5     EndIf
```

Les **mots clés** sont régulièrement décrits dans les chapitres sur le côté gauche de la page d'index du manuel de référence.

> Fonctions

Toutes les fonctions doivent avoir un nom suivi d'un '(' à défaut de quoi elle ne sera pas considérée comme une fonction. Cela est vrai y compris lorsque la fonction ne prend aucun paramètre.

Exemple

```
1  WindowEvent() ; est une fonction.
2  WindowEvent   ; est une variable.
```

> Constantes

Toutes les constantes ont un nom précédé par un #.
Elles ne peuvent être déclarées qu'une fois et garde toujours leur valeur prédéfinie. (Le compilateur remplace tous les noms des constantes avec leur valeur correspondante lors de la compilation de l'exécutable.)

Exemple

```
1  #Hello = 10 ; est une constante.
2  Hello  = 10 ; est une variable.
```

> Texte littéral

Les chaînes littérales sont encadrées par le caractère ". Les séquences d'échappement sont prises en charge en utilisant le caractère * * avant la chaîne littérale.
Les séquences d'échappement autorisés sont :

```
\a: bip                    Chr(7)
\b: retour arrière         Chr(8)
\f: saut de page           Chr(12)
\n: retour à la ligne      Chr(10)
\r: retour chariot         Chr(13)
\t: tabulation horizontal  Chr(9)
\v: tabulation vertical    Chr(11)
\": double quote           Chr(34)
\\: antislash              Chr(92)
```

Il y a deux constantes spéciales pour les chaînes :

```
#Empty$: représente une chaîne vide (exactement la même chose que "")
#Null$ : représente une chaîne nulle. Ceci peut être utilisé avec les
 fonctions
         des APIs nécessitant un pointeur NULL en guise de chaîne ou
 à une chaîne vraiment libre.
```

Exemple

```
1  a$ =  "Salut le monde"  ; Texte standard
2  b$ = ~"Echappe\nMoi !"  ; Texte avec une séquence d'échappement
```

> Labels

Tous les labels (étiquettes) doivent être suivis par un ' :'. Les noms de label ne peuvent pas contenir d'opérateurs (+,-,...) ou de caractères spéciaux (é,à,ß,ä,ö,ü,...).
Dans une procédure , l'étiquette sera disponible uniquement dans cette procédure.

Exemple

```
1  Je_suis_un_label:
```

> Expressions

On appelle expression toute séquence de code qui peut être évaluée. Une expression peut regrouper toute variable, constante ou fonction d'un même type. Lorsque vous utilisez des nombres dans une expression, vous pouvez utiliser le symbole $ en tête pour préciser qu'il s'agit d'une valeur hexadécimale ou un % pour signifier une valeur binaire. Sans l'un ou l'autre de ces deux symboles, la valeur sera toujours considérée comme décimale. Les chaînes de caractères doivent être délimitées par des guillemets.

Exemple

```
1   a = a + 1 + (12 * 3)
2   a = a + WindowHeight() + b/2 + #MaConstante
3
4   If a <> 12 + 2
5     b + 2 >= c + 3
6   EndIf
7
8   a.s = b.s + "ceci est une chaine" + c.s
9   a$ = b$ + "ceci est une autre chaine" + c$
10
11  foo = foo + $69 / %1001 ; Utilisation de nombres hexadécimal
     et binaire
```

> Regroupement des commandes

Il est possible de placer un nombre quelconque de commandes sur la même ligne en les séparant par :.

Exemple

```
1  If Variable=0 : Debug "Ok" : Else : Debug "Erreur" : EndIf
```

> Texte multiligne

Si une ligne de code contient une expression de grande taille, elle peut être divisée en plusieurs lignes. Une ligne découpée doit se terminer avec l'un des opérateurs suivants : plus (+), virgule (,), ou (||), And, Or, Xor.

Exemple

```
1  Texte$ = "Très très très très long texte" + #LF$ +
2           "un autre long texte" + #LF$ +
3           "et la fin du long texte."
4
5  MessageRequester("Bonjour c'est un titre très long.",
6                   "Et un très long message, afin que nous
    puissions utiliser le multiligne" + #LF$ + Texte$,
7                   #PB_MessageRequester_Ok)
```

> Glossaire

Les mots suivants utilisés dans ce manuel ont toujours le même sens :
<variable> : une variable basic.
<expression> : une expression comme commenté ci-dessus.
<constant> : une constante numérique.
<label> : un label de programme.
<type> : tout type, (standard ou structuré).

> Autres

- Dans ce manuel, tous les sujets sont listés en ordre alphabétique pour réduire tout temps de recherche.
- La **valeur renvoyée** par les commandes est le plus souvent un Integer . Dans le cas contraire, le type de la valeur est indiqué dans la description (ligne de syntaxe) de la commande.
- Dans la documentation de PureBasic, les termes "commandes" et "fonctions" ont le même sens, independamment du fait que la fonction retourne une valeur ou non.

Chapitre 34

Global

Syntax

```
Global[.<type>] <variable[.<type>]> [= <expression>] [, ...]
```

Description

Global permet à des variables d'être utilisées globalement, pouvant ainsi être accessibles depuis l'intérieur de n'importe quelle procédure. Une valeur par défaut peut être assignée à la variable. Global peut aussi être utilisé avec les tableaux , les listes et les maps .
Les instructions Protected et Static permettent de déclarer une variable locale dans une procédure qui a le même nom qu'une variable globale, sans risque de conflit.

Exemple

Exemple : Avec des variables

```
1  Global a.l, b.b, c, d = 20
2
3  Procedure Change()
4    Debug a  ; Affiche 10 car la variable 'a' est globale
5  EndProcedure
6
7  a = 10
8  Change()
```

Exemple : Avec un tableau

```
1  Global Dim Tableau(2)
2
3  Procedure Change()
4    Debug Tableau(0)  ; Affiche 10 car le tableau 'Tableau()' est global
5  EndProcedure
6
7  Tableau(0) = 10
8  Change()
```

Chapitre 35

Gosub : Return

Syntax

```
Gosub MonLabel

MonLabel:
  ...
Return
```

Description

Gosub signifie 'Go to sub routine', en français : 'Aller au sous-programme'. Un label doit être spécifié après Gosub pour que l'exécution du programme se poursuive à la position du label et jusqu'à ce qu'un Return soit rencontré. Lorsque le Return est atteint, le programme revient à l'instruction suivant le Gosub. Gosub est pratique pour construire rapidement un code structuré.
Les procédures sont une autre alternative pour la conception rapide d'un programme structuré. Gosub peut être seulement utilisé dans la partie principale du programme, pas dans les procedures .

Exemple

```
1   a = 1
2   b = 2
3   Gosub OperationComplexe
4   Debug a
5   End
6
7   OperationComplexe:
8     a = b*2+a*3+(a+b)
9     a = a+a*a
10  Return
```

Syntax

```
FakeReturn
```

Description

Lorsque vous souhaitez sauter d'un sous-programme à une autre partie du code extérieur au sous-programme (en utilisant une commande Goto), vous pouvez utiliser un FakeReturn qui simule un

Return sans l'effectuer réellement. Si vous n'utilisez pas ce dispositif, votre programme génèrera une erreur. Cette commande ne devrait pas avoir d'utilité car un programme bien conçu ne devrait pas utiliser de Goto . Toutefois, dans certains cas où la performance est critique, cela peut aider le programmeur.

Exemple

```
1  Gosub SousProgramme1
2
3  SousProgramme1:
4    ...
5    If a = 10
6      FakeReturn
7      Goto ProgrammePrincipal
8    EndIf
9  Return
```

Chapitre 36

Numéros et Identifiants (Handles)

Les numéros

Tous les objets crées sont identifiés par un numéro arbitraire (qui n'est pas l'identifiant de l'objet, comme défini ci-dessous). Dans ce manuel, les numéros s'écrivent sous la forme #Numéro (par exemple, chaque gadget créé a un numéro #Gadget).
Les numéros que vous leurs attribuez n'ont pas besoin d'être des constantes, mais ils doivent être uniques pour chaque objet de votre programme (une image peut avoir le même numéro qu'un gadget, parce que ce sont des types d'objets différents). Ces numéros sont utilisés pour accéder ultérieurement à ces objets dans votre programme.
Par exemple, les fonctions de gestion des évènements renvoient des numéros :

```
1  EventGadget()
2  EventMenu()
3  EventWindow()
```

Les identifiants

Tous les objets ont également un numéro unique qui leur est attribué par le système. Ces numéros uniques sont appelés identifiants. Parfois, une fonction PureBasic n'a pas besoin du numéro comme paramètre, mais de l'identifiant. Dans ce manuel, ce genre de besoin est indiqué, à l'aide du suffixe ID.

Exemple

```
1  ImageGadget(#Gadget, x, y, Largeur, Hauteur, ImageID [, Options])
2  ; #Gadget doit être le numéro que vous voulez attribuer au Gadget
3  ; ImageID doit être l'IDentifiant (numéro unique) de l'image.
```

Pour obtenir l'identifiant d'un objet, il y a des fonctions spéciales telles que :

```
1  FontID()
2  GadgetID()
3  ImageID()
4  ThreadID()
5  WindowID()
```

La plupart des fonctions qui créent ces objets renvoient cet identifiant comme résultat, si tout s'est bien passé. Ceci est seulement le cas si #PB_Any n'a pas été utilisé pour créer l'objet. Si #PB_Any est

utilisé, ces fonctions renvoient le numéro de l'objet qui a été attribué par PB pour elles, mais pas l'identifiant.

Exemple

```
1   GadgetHandle = ImageGadget(...)
```

Chapitre 37

If : Else : EndIf

Syntax

```
If <expression>
  ...
[ElseIf <expression>]
  ...
[Else]
  ...
EndIf
```

Description

La structure If est utilisée pour effectuer des tests et/ou changer le déroulement du programme selon le résultat (vrai ou faux) du test. ElseIf est utilisé pour produire un nombre quelconque de tests additionnels si le premier n'a pas eu un résultat vrai. La commande optionnelle Else est utilisée pour exécuter une séquence de code si tous les tests précédents de la structure ont échoué. Les structures If peuvent être imbriquées sans limite de profondeur.
Les court-circuits sont pris en charge, ce qui signifie que si un test est vrai, tous les tests suivants seront ignorés.

Exemple : Test simple

```
1  a = 5
2  If a = 10
3    Debug "a = 10"
4  Else
5    Debug "a <> 10"
6  EndIf
```

Exemple : Test multiple

```
1  b = 15
2  If a = 10 And b >= 10 Or c = 20
3    If b = 15
4      Debug "b = 15"
5    Else
6      PrintN("Autre possibilité")
7    EndIf
```

```
  Else
    PrintN("Erreur de test")
  EndIf
```

Exemple : Court-circuit

```
  Procedure AfficherSalut()
    Debug "Salut"
    ProcedureReturn 1
  EndProcedure

  a = 10
  If a = 10 Or AfficherSalut() = 1 ; a est égal à 10, alors le deuxième test est totalement ignoré
    Debug "Succès"
  Else
    Debug "Erreur"
  EndIf
```

Chapitre 38

Import : EndImport

Syntax

```
Import "NomFichier"
  NomFonction.<type>(<parametre>, [, <parametre> [= ValeurDefaut]...])
   [As "NomSymbole"]
  ...
  NomVariable.<type> [As "NomSymbole"]
EndImport
```

Description

Pour les programmeurs chevronnés. Import : EndImport permet de déclarer facilement des fonctions et des variables externes à partir d'un fichier bibliothèque (.lib) ou objet (.obj).
Une fois déclarées, les fonctions importées sont directement disponibles dans le programme, comme n'importe quelle autre commande. Le compilateur ne vérifie pas si la fonction existe réellement dans le fichier importé, donc si une erreur survient, elle sera reportée par le linker.
Les fonctions importées remplacent avantageusement les commandes OpenLibrary() /CallFunction() : la vérification du type, le nombre de paramètres sont validés par le compilateur. De plus, contrairement à CallFunction(), une fonction importée peut gérer les types 'double', 'float' et 'quad' sans aucun problème.
Les paramètres en fin de fonction peuvent avoir une valeur par défaut (une expression constante est requise). Les paramètres ayant une valeur par défaut pourront être omis lors de l'appel de la fonction, la valeur par défaut de chaque paramètre manquant sera utilisée.
Par défaut, le symbole de la fonction importée est décoré suivant le schéma suivant : _NomFonction@tailleargument. Cela devrait fonctionner pour la plupart des fonctions qui utilise la convention standard d'appel (stdcall). Par contre, si la bibliothèque est écrite en C, et que les fonctions exportées ne sont pas stdcall, ImportC devra être utilisé à la place de Import. Dans ce cas, par défaut le symbole est préfixé comme ceci : _NomFonction.
Les pseudotypes peuvent être utilisés pour les paramètres, mais pas pour le type de la valeur de retour.

Remarques

En 64 bits, il n'y a qu'une seule convention d'appel, alors ImportC fait la même chose que Import.

Exemple

```
1  Import "User32.lib"
2
```

```
    ; Pas besoin d'utiliser 'As' car PureBasic préfixe la fonction correctement
    ; Nous définissons également le paramètre 'Options' comme facultatif, avec une valeur par défaut de 0 (quand il est omis)
    ;
    MessageBoxA(Fenetre.i, Corps$, Titre$, Options.i = 0)

    ; Cette fois PureBasic ne peut pas se débrouiller tout seul, car le
    ; nom de la fonction n'est pas le même que celui utilisé par le symbole
    ;
    BoiteDeMessage(Fenetre.i, Corps$, Titre$, Options.i) As "_MessageBoxA@16"

EndImport

MessageBoxA(0, "Salut", "le Monde") ; Nous ne précisons pas les options
BoiteDeMessage(0, "Salut", "le Monde 2", 0)
```

Exemple : Avec les pseudotypes

```
Import "User32.lib"

    ; Nous utilisons le pseudotype 'p-unicode' pour les paramètres chaîne, car
    ; MessageBoxW() est une fonction unicode seulement. Le compilateur va
    ; automatiquement convertir les chaînes en unicode quand nécessaire.
    ;
    MessageBoxW(Fenetre.l, Corps.p-unicode, Titre.p-unicode, Options.l = 0)

EndImport

;
MessageBoxW(0, "Salut", "le Monde")
```

Chapitre 39

Les fonctions 'Include'

Syntax

```
IncludeFile  "NomFichier"
XIncludeFile "NomFichier"
```

Description

IncludeFile permet d'inclure tout fichier source spécifié à l'endroit où la commande est située dans le code même si XIncludeFile a été appelé avant.
Cette commande est utile, si vous voulez séparer votre code source en plusieurs fichiers, afin de réutiliser un même code dans différents projets.

Exemple

```
1  IncludeFile "Sources\myfile.pb" ; Ce fichier sera inséré à cet
     endroit du code.
```

Syntax

```
XIncludeFile "NomFichier"
```

Description

XIncludeFile est similaire à IncludeFile, mais en évitant toutefois d'inclure le même fichier plusieurs fois.
Cette commande est utile, si vous voulez séparer votre code source en plusieurs fichiers, afin de réutiliser un même code dans différents projets.

Exemple

```
1  XIncludeFile "Sources\monfichier.pb" ; monfichier.pb sera inséré.
2  XIncludeFile "Sources\monfichier.pb" ; monfichier.pb sera ignoré dans
     tout appel supplémentaire.
```

Syntax

```
IncludeBinary "NomFichier"
```

Description

IncludeBinary inclut le fichier nommé à l'endroit où la commande est placée.

Exemple

```
1   IncludeBinary "Data\map.data"
```

Syntax

```
IncludePath "Chemin"
```

Description

IncludePath spécifie le chemin d'accès par défaut pour tous les fichiers inclus appelés après la commande. Très pratique pour l'inclusion de nombreux fichiers provenant d'un même répertoire.

Exemple

```
1   IncludePath  "Sources\Data"
2   IncludeFile  "Sprite.pb"
3   XIncludeFile "Music.pb"
```

Chapitre 40

L'assembleur en ligne x86

PureBasic permet d'inclure toute commande assembleur x86 (y compris les instructions MMX et FPU) directement dans le code source comme dans un vrai source assembleur. Vous pouvez également utiliser directement les variables ou pointeurs avec les instructions assembleur et pouvez intégrer plusieurs commandes assembleur sur une même ligne.
Sous Windows et Linux, PureBasic utilise **fasm** (http ://flatassembler.net), alors pour plus d'information à propose de la syntaxe, lire le guide fasm.
Sous OS X, PureBasic utilise **yasm** (http ://yasm.tortall.net/), alors pour plus d'information à propose de la syntaxe, lire le guide yasm.
Pour utiliser l'assembleur en ligne, utiliser les directives du compilateur EnableASM ou DisableASM .
Il est possible d'activer la coloration syntaxique ASM dans l'IDE avec "Activer la coloration des mots clés assembleur" compiler option .

Règles

Vous devez respecter plusieurs règles précises pour inclure de l'assembleur dans du code PureBasic :
- Les variables et les pointeurs doivent être déclarés **préalablement** à leur utilisation dans un contexte assembleur.
- Concernant les étiquettes (labels) : Les labels référencés en assembleur sont toujours en minuscule. Par exemple, le label 'ETIquette :' sera transformé en 'l_etiquette' dans le code assembleur.
Lorsque vous faites référence à un label purebasic dans le corps du programme (en dehors des procedures et des modules) avec des fonctions assembleurs, vous devez précéder son nom par un L minuscule et d'un caractère de soulignement comme ceci **'l_'**.
Si le label est défini dans une procédure , alors son préfixe est **'ll_nomprocedure_'**, en minuscules (ll comme local label) mais il n'est accessible qu'à l'intérieur de la procédure.
Pour accéder à un label dans un module avec des fonctions assembleurs, vous devez ajouter devant le label le préfixe **'nomdumodule.l_nomprocedure_'** écrit tout en minuscule.
Et enfin un label défini dans une procédure elle-même dans un module s'écrira **'nomdumodule.ll_nomprocedure_'** - Pour info, les listes commencent par 't_nomlist', les map par 'm_nommap' et les tableaux par 'a_nomtableau' est toujours en minuscule.

Exemple

```
1  DeclareModule MonModule
2    LabelDeclareModule: ;Son nom assembleur est
     monmodule.l_labeldeclaremodule:
3    Declare Init()
4  EndDeclareModule
5
6  Module MonModule
```

```
7     Procedure Init()
8       LabelModuleProcedure: ; Son nom assembleur est
      monmodule.ll_init_labelmoduleprocedure:
9       Debug "InitFerrari()"
10    EndProcedure
11
12    LabelModule1: ;Son nom assembleur est monmodule.l_labelmodule1:
13  EndModule
14
15   Procedure Test (*Pointer, Variable)
16    TokiSTART:  ;Son nom assembleur est ll_test_tokistart:
17
18    ! MOV dword [p.p_Pointer], 20
19    ! MOV dword [p.v_Variable], 30
20    Debug *Pointer  ;Son nom assembleur est p.p_Pointer
21    Debug Variable  ;Son nom assembleur est p.v_Variable
22  EndProcedure
23
24  VAR=1                         ;Son nom assembleur est v_VAR
25  *Pointt=AllocateMemory(10)  ;Son nom assembleur est p_Pointt
26
27  MonModule::Init()
28  Test(0,0)
29
30  Label1: ;Son nom assembleur est l_label1:
31  !jmp l_labelend ; Une instruction en assembler doit suivre les règles
     ci-dessus. Ici c'est l_nomelabel
32  ;...
33  LabelEnd: ;Son nom assembleur est l_labelend:
```

- Les erreurs dans une section asm ne sont pas reportées par PureBasic mais par Fasm. Vérifiez votre code si une telle erreur survient.
- Avec l'assembleur en ligne activé, vous ne pouvez pas utiliser les mots clés ASM pour les étiquettes (label).
- **Sur les processeurs x86, les registres volatiles sont : eax, ecx, edx, xmm0, xmm1, xmm2 et xmm3.** Tous les autres doivent être préservés.
- **Sur les processeurs x64, les registres volatiles sont : rax, rcx, rdx, r8, r9, xmm0, xmm1, xmm2 et xmm3.** Tous les autres doivent être préservés.
- Windows : un fichier aide-ASM peut être téléchargé ici. Si vous déplacez le fichier 'ASM.HLP' dans le dossier 'Help/' de PureBasic, vous aurez accès à l'aide sur les mots clés assembleur en appuyant sur F1. (Note : Ne fonctionne que si l'option 'assembleur en ligne' est activé).

Quand on utilise l'assembleur dans une procédure , il est utile de connaître les points suivants :
- Pour renvoyer directement la valeur du registre 'eax' (ou 'rax' sur x64)comme valeur de retour, il suffit d'utiliser ProcedureReturn, sans paramètre. Le contenu du registre 'eax' (ou 'rax' sur x64) restera inchangé et sera utilisé comme valeur de retour.

Exemple

```
1   Procedure.l MonTest()
2     !MOV eax, 45
3     ProcedureReturn  ; La valeur de retour sera 45
4   EndProcedure
5   Debug MonTest()
```

- Les variables locales en PureBasic sont directement indexées par rapport au registre de la **pile** (ESP) ce qui implique qu'un changement de la valeur de ce registre par une instruction assembleur (tel que PUSH, POP etc..) la référence vers la variable ne sera plus correcte.

- Il est possible de passer directement une ligne complète à l'assembleur sans aucune modification en utilisant le caractère **'!'** en début de ligne. Ceci permet d'avoir un accès total aux fonctionnalités de l'assembleur. Pour faciliter l'accès aux variables locales, une notation a été mise en place : **'p.v_NomVariable'** pour une variable standard et **'p.p_NomPointeur'** pour un pointeur.

Exemple

```
1  Procedure Test(*Pointer, Variable)
2    ! MOV dword [p.p_Pointer], 20
3    ! MOV dword [p.v_Variable], 30
4    Debug *Pointer
5    Debug Variable
6  EndProcedure
7
8  Test(0, 0)
```

Chapitre 41

Interfaces

Syntax

```
Interface <nom> [Extends <nom>]
  <Methode[.<type>]()>
...
EndInterface
```

Description

Les Interfaces sont utilisées pour accéder facilement aux modules 'Orientés Objets' tels que les bibliothèques COM (Component Object Model) ou DirectX. Ce type de bibliothèques sont de plus en plus courantes sous Windows et les interfaces permettent une exploitation de ces fonctions sans impacts de performances. Les interfaces jettent aussi les bases pour une 'Programmation Orientée Object' (OOP en anglais) avec PureBasic mais de solides connaissances sont nécessaires pour en tirer parti (les interfaces n'ont pas été conçues pour ajouter une couche objet à PureBasic mais plutôt pour accéder facilement à des objets dejà conçus). La plupart des interfaces utilisées sous Windows sont déjà incluses dans les fichiers résidents 'Interfaces.res' et 'InterfaceDX.res', ce qui rend leur utilisation immédiate.
Le paramètre optionnel Extends permet d'étendre une interface sans avoir à dupliquer ses fonctions (ces 'fonctions' sont aussi communément appelées 'méthodes' dans les autres langages objet tels que C++ ou Java). Toutes les fonctions contenues dans l'interface étendue seront disponibles dans la nouvelle interface. C'est utile pour un héritage simple d'objets.
les tableaux (Arrays) peuvent être passés en paramètres à l'aide du mot clé Array, Les listes en utilisant le mot clé List et les maps en utilisant le mot clé Map.
Un type de retour peut être défini dans la déclaration de l'interface en ajoutant le type après la méthode.
La commande SizeOf peut être utilisée avec les interfaces pour déterminer la taille d'une interface et la commande OffsetOf peut être utilisée pour déterminer l'index d'une fonction dans une interface.
Les pseudotypes peuvent être utilisés pour les paramètres des fonctions de l'interface, mais pas pour le type de retour.
Note : Les concepts objets sont principalement appropriés pour les développeurs expérimentés et il n'est pas nécessaire de les maîtriser ou même les comprendre pour réaliser des applications ou jeux professionnels.

Exemple : Appel à une fonction objet

```
1  ; Nous allons considérer que vous voulez accéder à un objet externe
     (à l'intérieur d'une DLL par exemple).
2  ; Premièrement, déclarez son interface.
3  ;
4  Interface MonObjet
5    Deplacer(x,y)
```

```
    DeplacerF(x.f,y.f)
    Detruire()
  EndInterface

  ; Si 'CreationObjet()' est la fonction qui crée l'objet, à partir de la DLL,
  ; dont l'interface vient d'être définie...
  ; Alors créons le premier objet.
  ;
  Objet1.MonObjet = CreationObjet()

  ; Et le deuxième.
  ;
  Objet2.MonObjet = CreationObjet()

  ; Ensuite, les fonctions qui viennent d'être définies, peuvent être utilisées,
  ; afin d'agir sur l'objet désiré.

  Objet1\Deplacer(10, 20)
  Objet1\Detruire()

  Objet2\DeplacerF(10.5, 20.1)
  Objet2\Detruire()
```

Exemple : Utilisation de 'Extends'

```
  ; Définition d'une interface générique 'Cube'
  ;
  Interface Cube
    EnvoyerPosition()
    DefinirPosition(x)
    EnvoyerLargeur()
    DefinirLargeur(Largeur)
  EndInterface

  Interface CubeColorer Extends Cube
    EnvoyerCouleur()
    DefinirCouleur(Couleur)
  EndInterface

  Interface CubeTexturer Extends Cube
    EnvoyerTexture()
    DefinirTexture(TextureID)
  EndInterface

  ; Nous avons maintenant 3 interfaces pour chaque objet:
  ;
  ; - 'Cube' a les fonctions Envoyer/DefinirPosition() et Envoyer/DefinirLargeur()
  ; - 'CubeColorer' a les fonctions Envoyer/DefinirPosition(), Envoyer/DefinirLargeur() et Envoyer/DefinirCouleur()
  ; - 'CubeTexturer' a les fonctions Envoyer/DefinirPosition(),Envoyer/DefinirLargeur() et Envoyer/DefinirTexture()
  ;
```

Chapitre 42

Licenses for the PureBasic applications (without using 3D engine)

```
This program makes use of the following components:

----------------------------------------------------------------------
Component: MD5
----------------------------------------------------------------------

Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.

----------------------------------------------------------------------
Component: AES
----------------------------------------------------------------------

Optimized ANSI C code for the Rijndael cipher (now AES)

@authorVincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>
@authorAntoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>
@authorPaulo Barreto <paulo.barreto@terra.com.br>

This code is hereby placed in the public domain.
```

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; *LOSS OF USE, DATA, OR PROFITS; OR* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: SHA1

SHA-1 in C
By Steve Reid <steve@edmweb.com>
100% Public Domain

Component: zlib

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; *you must not* claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly — jloup@gzip.org
Mark Adler — madler@alumni.caltech.edu

Component: libpq

Portions Copyright (c) 1996-2011, PostgreSQL Global Development Group
Portions Copyright (c) 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement
is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

```
IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
   FOR
DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES,
   INCLUDING
LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS
DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF
   THE
POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER
   IS
ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO
   OBLIGATIONS TO
PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

--------------------------------------------------------------------
Component: sqlite3
--------------------------------------------------------------------

The author disclaims copyright to this source code.  In place of
a legal notice, here is a blessing:

    May you do good and not evil.
    May you find forgiveness for yourself and forgive others.
    May you share freely, never taking more than you give.

--------------------------------------------------------------------
Component: libjpeg
--------------------------------------------------------------------

The authors make NO WARRANTY or representation, either express or
   implied,
with respect to this software, its quality, accuracy, merchantability,
   or
fitness for a particular purpose.  This software is provided "AS IS",
   and you,
its user, assume the entire risk as to its quality and accuracy.

This software is copyright (C) 1991-2012, Thomas G. Lane, Guido
   Vollbeding.
All Rights Reserved except as specified below.

Permission is hereby granted to use, copy, modify, and distribute this
software (or portions thereof) for any purpose, without fee, subject to
   these
conditions:
(1) If any part of the source code for this software is distributed,
   then this
README file must be included, with this copyright and no-warranty notice
unaltered; and any additions, deletions, or changes to the original
   files
must be clearly indicated in accompanying documentation.
(2) If only executable code is distributed, then the accompanying
documentation must state that "this software is based in part on the
   work of
the Independent JPEG Group".
```

(3) Permission for use of this software is granted only if the user accepts
full responsibility for any undesirable consequences; the authors accept
NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code,
not just to the unmodified library. If you use our work, you ought to
acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name
in advertising or publicity relating to this software or products derived from
it. This software may be referred to only as "the Independent JPEG Group's
software".

We specifically permit and encourage the use of this software as the basis of
commercial products, provided that all warranty or liability claims are
assumed by the product vendor.

Component: libpng
---

libpng versions 1.2.6, August 15, 2004, through 1.5.12, July 11, 2012, are
Copyright (c) 2004, 2006-2012 Glenn Randers-Pehrson, and are
distributed according to the same disclaimer and license as libpng-1.2.5
with the following individual added to the list of Contributing Authors:

   Cosmin Truta

libpng versions 1.0.7, July 1, 2000, through 1.2.5, October 3, 2002, are
Copyright (c) 2000-2002 Glenn Randers-Pehrson, and are
distributed according to the same disclaimer and license as libpng-1.0.6
with the following individuals added to the list of Contributing Authors:

   Simon-Pierre Cadieux
   Eric S. Raymond
   Gilles Vollant

and with the following additions to the disclaimer:

   There is no warranty against interference with your enjoyment of the
   library or against infringement. There is no warranty that our
   efforts or the library will fulfill any of your particular purposes
   or needs. This library is provided with all faults, and the entire
   risk of satisfactory quality, performance, accuracy, and effort is with
   the user.

libpng versions 0.97, January 1998, through 1.0.6, March 20, 2000, are
Copyright (c) 1998, 1999, 2000 Glenn Randers-Pehrson, and are
distributed according to the same disclaimer and license as libpng-0.96,
with the following individuals added to the list of Contributing

```
   Authors:

   Tom Lane
   Glenn Randers-Pehrson
   Willem van Schaik

libpng versions 0.89, June 1996, through 0.96, May 1997, are
Copyright (c) 1996, 1997 Andreas Dilger
Distributed according to the same disclaimer and license as libpng-0.88,
with the following individuals added to the list of Contributing
   Authors:

   John Bowler
   Kevin Bracey
   Sam Bushell
   Magnus Holmgren
   Greg Roelofs
   Tom Tanner

libpng versions 0.5, May 1995, through 0.88, January 1996, are
Copyright (c) 1995, 1996 Guy Eric Schalnat, Group 42, Inc.

For the purposes of this copyright and license, "Contributing Authors"
is defined as the following set of individuals:

   Andreas Dilger
   Dave Martindale
   Guy Eric Schalnat
   Paul Schmidt
   Tim Wegner

The PNG Reference Library is supplied "AS IS".  The Contributing Authors
and Group 42, Inc. disclaim all warranties, expressed or implied,
including, without limitation, the warranties of merchantability and of
fitness for any purpose.  The Contributing Authors and Group 42, Inc.
assume no liability for direct, indirect, incidental, special,
   exemplary,
or consequential damages, which may result from the use of the PNG
Reference Library, even if advised of the possibility of such damage.

Permission is hereby granted to use, copy, modify, and distribute this
source code, or portions hereof, for any purpose, without fee, subject
to the following restrictions:

  1. The origin of this source code must not be misrepresented.

  2. Altered versions must be plainly marked as such and must not
     be misrepresented as being the original source.

  3. This Copyright notice may not be removed or altered from
     any source or altered source distribution.

The Contributing Authors and Group 42, Inc. specifically permit, without
fee, and encourage the use of this source code as a component to
supporting the PNG file format in commercial products.  If you use this
source code in a product, acknowledgment is not required but would be
appreciated.

------------------------------------------------------------------------
```

```
Component: OpenJPEG
-----------------------------------------------------------------------

Copyright (c) 2002-2007, Communications and Remote Sensing Laboratory,
   Universite catholique de Louvain (UCL), Belgium
Copyright (c) 2002-2007, Professor Benoit Macq
Copyright (c) 2001-2003, David Janssens
Copyright (c) 2002-2003, Yannick Verschueren
Copyright (c) 2003-2007, Francois-Olivier Devaux and Antonin Descampe
Copyright (c) 2005, Herve Drolon, FreeImage Team
Copyright (c) 2006-2007, Parvatha Elangovan
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS
   IS'
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
   THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
   PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
   BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
   THE
POSSIBILITY OF SUCH DAMAGE.

-----------------------------------------------------------------------
Component: libtiff
-----------------------------------------------------------------------

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

Permission to use, copy, modify, distribute, and sell this software and
its documentation for any purpose is hereby granted without fee,
   provided
that (i) the above copyright notices and this permission notice appear
   in
all copies of the software and related documentation, and (ii) the
   names of
Sam Leffler and Silicon Graphics may not be used in any advertising or
publicity relating to the software without the specific, prior written
permission of Sam Leffler and Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
```

```
WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR
ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND,
OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY
   OF
LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.

-----------------------------------------------------------------------
Component: libmodplug (Module)
-----------------------------------------------------------------------

This source code is public domain.

-----------------------------------------------------------------------
Component: udis86 (OnError)
-----------------------------------------------------------------------

Copyright (c) 2002-2009 Vivek Thampi
All rights reserved.

Redistribution and use in source and binary forms, with or without
   modification,
are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
   notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
   notice,
      this list of conditions and the following disclaimer in the
   documentation
      and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
   IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
   LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
   DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
   SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
   AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
   OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

-----------------------------------------------------------------------
Component: brieflz
-----------------------------------------------------------------------

Copyright (c) 2002-2004 by Joergen Ibsen / Jibz
```

```
All Rights Reserved

http://www.ibsensoftware.com/

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

 1. The origin of this software must not be misrepresented; you must
    not claim that you wrote the original software. If you use this
    software in a product, an acknowledgment in the product
    documentation would be appreciated but is not required.

 2. Altered source versions must be plainly marked as such, and must
    not be misrepresented as being the original software.

 3. This notice may not be removed or altered from any source
    distribution.

-------------------------------------------------------------------
Component: jcalg1
-------------------------------------------------------------------

This software is provided as-is, without warranty of ANY KIND, either
   expressed
or implied, including but not limited to the implied warranties of
   merchantability
and/or fitness for a particular purpose. The author shall NOT be held
   liable for
ANY damage to you, your computer, or to anyone or anything else, that
   may result
from its use, or misuse. Basically, you use it at YOUR OWN RISK.

-------------------------------------------------------------------
Component: lzma
-------------------------------------------------------------------

LZMA SDK is written and placed in the public domain by Igor Pavlov.

Some code in LZMA SDK is based on public domain code from another
   developers:
  1) PPMd var.H (2001): Dmitry Shkarin
  2) SHA-256: Wei Dai (Crypto++ library)

-------------------------------------------------------------------
Component: libzip
-------------------------------------------------------------------

Copyright (C) 1999-2008 Dieter Baron and Thomas Klausner
The authors can be contacted at <libzip@nih.at>

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
```

notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: pcre

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

* Neither the name of the University of Cambridge nor the name of Google
Inc. nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

POSSIBILITY OF SUCH DAMAGE.

Component: scintilla

License for Scintilla and SciTE

Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
OR PERFORMANCE OF THIS SOFTWARE.

Component: expat

Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                                and Clark Cooper
Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006 Expat maintainers.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Component: libogg

Copyright (c) 2002, Xiph.org Foundation

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: libvorbis

Copyright (c) 2002-2004 Xiph.org Foundation

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE

```
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

-----------------------------------------------------------------------
Component: neuquant
-----------------------------------------------------------------------

NeuQuant Neural-Net Quantization Algorithm Interface
-----------------------------------------------------

Copyright (c) 1994 Anthony Dekker

NEUQUANT Neural-Net quantization algorithm by Anthony Dekker, 1994.
See "Kohonen neural networks for optimal colour quantization"
in "Network: Computation in Neural Systems" Vol. 5 (1994) pp 351-367.
for a discussion of the algorithm.
See also  http://members.ozemail.com.au/~dekker/NEUQUANT.HTML

Any party obtaining a copy of these files from the author, directly or
indirectly, is granted, free of charge, a full and unrestricted
   irrevocable,
world-wide, paid up, royalty-free, nonexclusive right and license to
   deal
in this software and documentation files (the "Software"), including
   without
limitation the rights to use, copy, modify, merge, publish, distribute,
   sublicense,
and/or sell copies of the Software, and to permit persons who receive
copies from any such party to do so, with the only requirement being
that this copyright notice remain intact.


Modified to quantize 32bit RGBA images for the pngnq program.
Also modified to accept a numebr of colors arguement.
Copyright (c) Stuart Coyle 2004-2006



Rewritten by Kornel LesiÅski (2009)
Euclidean distance, color matching dependent on alpha channel
and with gamma correction. code refreshed for modern
   compilers/architectures:
ANSI C, floats, removed pointer tricks and used arrays and structs.
```

Chapitre 43

Licenses for the 3D engine integrated with PureBasic

```
This program makes use of the following components:

-------------------------------------------------------------------------
Component: OGRE
-------------------------------------------------------------------------

OGRE (www.ogre3d.org) is made available under the MIT License.

Copyright (c) 2000-2012 Torus Knot Software Ltd

Permission is hereby granted, free of charge, to any person obtaining a
    copy
of this software and associated documentation files (the "Software"),
    to deal
in the Software without restriction, including without limitation the
    rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
    sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
    in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
    OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
    THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
    IN
THE SOFTWARE.

-------------------------------------------------------------------------
Component: CEGUI
```

```
------------------------------------------------------------------------

Copyright (C) 2004 - 2006 Paul D Turner & The CEGUI Development Team

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.

------------------------------------------------------------------------
Component: bullet
------------------------------------------------------------------------

Copyright (c) 2003-2010 Erwin Coumans
   http://continuousphysics.com/Bullet/

This software is provided 'as-is', without any express or implied
   warranty.
In no event will the authors be held liable for any damages arising
   from the use of this software.
Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
   freely,
subject to the following restrictions:

1. The origin of  this software must not  be misrepresented; you must
   not claim
that you wrote the original software. If you use this software in a
   product,  an
acknowledgment in  the product  documentation would  be appreciated
   but is  not
required.

2. Altered source  versions must be plainly  marked as such, and  must
not be misrepresented as being the original software.

3. This notice may not be removed or altered from any source
   distribution.

------------------------------------------------------------------------
Component: FreeImage
------------------------------------------------------------------------

FreeImage Public License - Version 1.0
```

1. Definitions.

1.1. "Contributor" means each entity that creates or contributes to the creation
of Modifications.

1.2. "Contributor Version" means the combination of the Original Code, prior
Modifications used by a Contributor, and the Modifications made by that
particular Contributor.

1.3. "Covered Code" means the Original Code or Modifications or the combination
of the Original Code and Modifications, in each case including portions thereof.

1.4. "Electronic Distribution Mechanism" means a mechanism generally accepted in
the software development community for the electronic transfer of data.

1.5. "Executable" means Covered Code in any form other than Source Code.

1.6. "Initial Developer" means the individual or entity identified as the
Initial Developer in the Source Code notice required by Exhibit A.

1.7. "Larger Work" means a work which combines Covered Code or portions thereof
with code not governed by the terms of this License.

1.8. "License" means this document.

1.9. "Modifications" means any addition to or deletion from the substance or
structure of either the Original Code or any previous Modifications. When
Covered Code is released as a series of files, a Modification is:

A. Any addition to or deletion from the contents of a file containing Original
Code or previous Modifications.

B. Any new file that contains any part of the Original Code or previous
Modifications.

1.10. "Original Code" means Source Code of computer software code which is
described in the Source Code notice required by Exhibit A as Original Code, and
which, at the time of its release under this License is not already Covered Code
governed by this License.

1.11. "Source Code" means the preferred form of the Covered Code for making
modifications to it, including all modules it contains, plus any associated

interface definition files, scripts used to control compilation and installation
of an Executable, or a list of source code differential comparisons against
either the Original Code or another well known, available Covered Code of the
Contributor's choice. The Source Code can be in a compressed or archival form,
provided the appropriate decompression or de-archiving software is widely
available for no charge.

1.12. "You" means an individual or a legal entity exercising rights under, and
complying with all of the terms of, this License or a future version of this
License issued under Section 6.1. For legal entities, "You" includes any entity
which controls, is controlled by, or is under common control with You. For
purposes of this definition, "control" means (a) the power, direct or indirect,
to cause the direction or management of such entity, whether by contract or
otherwise, or (b) ownership of fifty percent (50%) or more of the outstanding
shares or beneficial ownership of such entity.

2. Source Code License.

2.1. The Initial Developer Grant. The Initial Developer hereby grants You a
world-wide, royalty-free, non-exclusive license, subject to third party
intellectual property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the
Original Code (or portions thereof) with or without Modifications, or as part of
a Larger Work; and

(b) under patents now or hereafter owned or controlled by Initial Developer, to
make, have made, use and sell ("Utilize") the Original Code (or portions
thereof), but solely to the extent that any such patent is reasonably necessary
to enable You to Utilize the Original Code (or portions thereof) and not to any
greater extent that may be necessary to Utilize further Modifications or
combinations.

2.2. Contributor Grant. Each Contributor hereby grants You a world-wide,
royalty-free, non-exclusive license, subject to third party intellectual
property claims:

(a) to use, reproduce, modify, display, perform, sublicense and distribute the
Modifications created by such Contributor (or portions thereof) either on an
unmodified basis, with other Modifications, as Covered Code or as part of a
Larger Work; and

(b) under patents now or hereafter owned or controlled by Contributor, to
Utilize the Contributor Version (or portions thereof), but solely to the extent
that any such patent is reasonably necessary to enable You to Utilize the
Contributor Version (or portions thereof), and not to any greater extent that
may be necessary to Utilize further Modifications or combinations.

3. Distribution Obligations.

3.1. Application of License. The Modifications which You create or to which You
contribute are governed by the terms of this License, including without
limitation Section 2.2. The Source Code version of Covered Code may be
distributed only under the terms of this License or a future version of this
License released under Section 6.1, and You must include a copy of this License
with every copy of the Source Code You distribute. You may not offer or impose
any terms on any Source Code version that alters or restricts the applicable
version of this License or the recipients' rights hereunder. However, You may
include an additional document offering the additional rights described in
Section 3.5.

3.2. Availability of Source Code. Any Modification which You create or to which
You contribute must be made available in Source Code form under the terms of
this License either on the same media as an Executable version or via an
accepted Electronic Distribution Mechanism to anyone to whom you made an
Executable version available; *and if made available via Electronic Distribution*
Mechanism, must remain available for at least twelve (12) months after the date
it initially became available, or at least six (6) months after a subsequent
version of that particular Modification has been made available to such
recipients. You are responsible for ensuring that the Source Code version

remains available even if the Electronic Distribution Mechanism is maintained by
a third party.

3.3. Description of Modifications. You must cause all Covered Code to which you
contribute to contain a file documenting the changes You made to create that
Covered Code and the date of any change. You must include a prominent statement
that the Modification is derived, directly or indirectly, from Original Code
provided by the Initial Developer and including the name of the Initial
Developer in (a) the Source Code, and (b) in any notice in an Executable version
or related documentation in which You describe the origin or ownership of the
Covered Code.

3.4. Intellectual Property Matters

(a) Third Party Claims. If You have knowledge that a party claims an
intellectual property right in particular functionality or code (or its
utilization under this License), you must include a text file with the source
code distribution titled "LEGAL" which describes the claim and the party making
the claim in sufficient detail that a recipient will know whom to contact. If
you obtain such knowledge after You make Your Modification available as
described in Section 3.2, You shall promptly modify the LEGAL file in all copies
You make available thereafter and shall take other steps (such as notifying
appropriate mailing lists or newsgroups) reasonably calculated to inform those
who received the Covered Code that new knowledge has been obtained.

(b) Contributor APIs. If Your Modification is an application programming
interface and You own or control patents which are reasonably necessary to
implement that API, you must also include this information in the LEGAL file.

3.5. Required Notices. You must duplicate the notice in Exhibit A in each file
of the Source Code, and this License in any documentation for the Source Code,
where You describe recipients' rights relating to Covered Code. If You created
one or more Modification(s), You may add your name as a Contributor to the
notice described in Exhibit A. If it is not possible to put such notice in a

particular Source Code file due to its structure, then you must include such
notice in a location (such as a relevant directory file) where a user would be
likely to look for such a notice. You may choose to offer, and to charge a fee
for, warranty, support, indemnity or liability obligations to one or more
recipients of Covered Code. However, You may do so only on Your own behalf, and
not on behalf of the Initial Developer or any Contributor. You must make it
absolutely clear than any such warranty, support, indemnity or liability
obligation is offered by You alone, and You hereby agree to indemnify the
Initial Developer and every Contributor for any liability incurred by the
Initial Developer or such Contributor as a result of warranty, support,
indemnity or liability terms You offer.

3.6. Distribution of Executable Versions. You may distribute Covered Code in
Executable form only if the requirements of Section 3.1-3.5 have been met for
that Covered Code, and if You include a notice stating that the Source Code
version of the Covered Code is available under the terms of this License,
including a description of how and where You have fulfilled the obligations of
Section 3.2. The notice must be conspicuously included in any notice in an
Executable version, related documentation or collateral in which You describe
recipients' rights relating to the Covered Code. You may distribute the
Executable version of Covered Code under a license of Your choice, which may
contain terms different from this License, provided that You are in compliance
with the terms of this License and that the license for the Executable version
does not attempt to limit or alter the recipient's rights in the Source Code
version from the rights set forth in this License. If You distribute the
Executable version under a different license You must make it absolutely clear
that any terms which differ from this License are offered by You alone, not by
the Initial Developer or any Contributor. You hereby agree to indemnify the
Initial Developer and every Contributor for any liability incurred by the
Initial Developer or such Contributor as a result of any such terms You offer.

3.7. Larger Works. You may create a Larger Work by combining Covered Code with
other code not governed by the terms of this License and distribute the Larger
Work as a single product. In such a case, You must make sure the requirements of
this License are fulfilled for the Covered Code.

4. Inability to Comply Due to Statute or Regulation.

If it is impossible for You to comply with any of the terms of this License with
respect to some or all of the Covered Code due to statute or regulation then You
must: (a) comply with the terms of this License to the maximum extent possible;
and (b) describe the limitations and the code they affect. Such description must
be included in the LEGAL file described in Section 3.4 and must be included with
all distributions of the Source Code. Except to the extent prohibited by statute
or regulation, such description must be sufficiently detailed for a recipient of
ordinary skill to be able to understand it.

5. Application of this License.

This License applies to code to which the Initial Developer has attached the
notice in Exhibit A, and to related Covered Code.

6. Versions of the License.

6.1. New Versions. Floris van den Berg may publish revised and/or new versions
of the License from time to time. Each version will be given a distinguishing
version number.

6.2. Effect of New Versions. Once Covered Code has been published under a
particular version of the License, You may always continue to use it under the
terms of that version. You may also choose to use such Covered Code under the
terms of any subsequent version of the License published by Floris van den Berg
No one other than Floris van den Berg has the right to modify the terms
applicable to Covered Code created under this License.

6.3. Derivative Works. If you create or use a modified version of this License
(which you may only do in order to apply it to code which is not already Covered
Code governed by this License), you must (a) rename Your license so that the
phrases "FreeImage", ‘FreeImage Public License", "FIPL", or any

confusingly
similar phrase do not appear anywhere in your license and (b) otherwise make it
clear that your version of the license contains terms which differ from the
FreeImage Public License. (Filling in the name of the Initial Developer,
Original Code or Contributor in the notice described in Exhibit A shall not of
themselves be deemed to be modifications of this License.)

7. DISCLAIMER OF WARRANTY.

COVERED CODE IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT
WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT
LIMITATION, WARRANTIES THAT THE COVERED CODE IS FREE OF DEFECTS, MERCHANTABLE,
FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE
QUALITY AND PERFORMANCE OF THE COVERED CODE IS WITH YOU. SHOULD ANY COVERED CODE
PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL DEVELOPER OR ANY OTHER
CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION.
THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO
USE OF ANY COVERED CODE IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. TERMINATION.

This License and the rights granted hereunder will terminate automatically if
You fail to comply with terms herein and fail to cure such breach within 30 days
of becoming aware of the breach. All sublicenses to the Covered Code which are
properly granted shall survive any termination of this License. Provisions
which, by their nature, must remain in effect beyond the termination of this
License shall survive.

9. LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER TORT (INCLUDING
NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL DEVELOPER, ANY OTHER
CONTRIBUTOR, OR ANY DISTRIBUTOR OF COVERED CODE, OR ANY SUPPLIER OF ANY OF SUCH
PARTIES, BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL,
INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT
LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER

FAILURE OR
MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF SUCH
PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES. THIS
LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL
INJURY RESULTING FROM SUCH PARTY'S NEGLIGENCE TO THE EXTENT APPLICABLE LAW
PROHIBITS SUCH LIMITATION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR
LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THAT EXCLUSION AND
LIMITATION MAY NOT APPLY TO YOU.

10. U.S. GOVERNMENT END USERS.

The Covered Code is a "commercial item," as that term is defined in 48 C.F.R.
2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial
computer software documentation," as such terms are used in 48 C.F.R. 12.212
(Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through
227.7202-4 (June 1995), all U.S. Government End Users acquire Covered Code with
only those rights set forth herein.

11. MISCELLANEOUS.

This License represents the complete agreement concerning subject matter hereof.
If any provision of this License is held to be unenforceable, such provision
shall be reformed only to the extent necessary to make it enforceable. This
License shall be governed by Dutch law provisions (except to the extent
applicable law, if any, provides otherwise), excluding its conflict-of-law
provisions. With respect to disputes in which at least one party is a citizen
of, or an entity chartered or registered to do business in, the The Netherlands:
(a) unless otherwise agreed in writing, all disputes relating to this License
(excepting any dispute relating to intellectual property rights) shall be
subject to final and binding arbitration, with the losing party paying all costs
of arbitration; *(b) any arbitration relating to this Agreement shall be held in*
Almelo, The Netherlands; *and (c) any litigation relating to this Agreement shall*
be subject to the jurisdiction of the court of Almelo, The Netherlands with the
losing party responsible for costs, including without limitation, court costs

and reasonable attorneys fees and expenses. Any law or regulation which provides
that the language of a contract shall be construed against the drafter shall not
apply to this License.

12. RESPONSIBILITY FOR CLAIMS.

Except in cases where another Contributor has failed to comply with Section 3.4,
You are responsible for damages arising, directly or indirectly, out of Your
utilization of rights under this License, based on the number of copies of
Covered Code you made available, the revenues you received from utilizing such
rights, and other relevant factors. You agree to work with affected parties to
distribute responsibility on an equitable basis.

EXHIBIT A.

"The contents of this file are subject to the FreeImage Public License Version
1.0 (the "License"); you may not use this file except in compliance with the
License. You may obtain a copy of the License at
http://home.wxs.nl/~flvdberg/freeimage-license.txt

Software distributed under the License is distributed on an "AS IS" basis,
WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the
specific language governing rights and limitations under the License.

Component: FreeType

The FreeType Project LICENSE

2006-Jan-27

Copyright 1996-2002, 2006 by
David Turner, Robert Wilhelm, and Werner Lemberg

Introduction
============

The FreeType Project is distributed in several archive packages;
some of them may contain, in addition to the FreeType font engine,
various tools and contributions which rely on, or relate to, the
FreeType Project.

This license applies to all files found in such packages, and

which do not fall under their own explicit license. The license affects thus the FreeType font engine, the test programs, documentation and makefiles, at the very least.

This license was inspired by the BSD, Artistic, and IJG (Independent JPEG Group) licenses, which all encourage inclusion and use of free software in commercial and freeware products alike. As a consequence, its main points are that:

- We don't promise that this software works. However, we will be interested in any kind of bug reports. ('as is' distribution)
- You can use this software for whatever you want, in parts or full form, without having to pay us. ('royalty-free' usage)
- You may not pretend that you wrote this software. If you use it, or only parts of it, in a program, you must acknowledge somewhere in your documentation that you have used the FreeType code. ('credits')

We specifically permit and encourage the inclusion of this software, with or without modifications, in commercial products. We disclaim all warranties covering The FreeType Project and assume no liability related to The FreeType Project.

Finally, many people asked us for a preferred form for a credit/disclaimer to use in compliance with this license. We thus encourage you to use the following text:

```
"""
 Portions of this software are copyright © <year> The FreeType
 Project (www.freetype.org).  All rights reserved.
"""
```

Please replace <year> with the value from the FreeType version you actually use.

Legal Terms

0. Definitions

Throughout this license, the terms 'package', 'FreeType Project', and 'FreeType archive' refer to the set of files originally distributed by the authors (David Turner, Robert Wilhelm, and Werner Lemberg) as the 'FreeType Project', be they named as alpha, beta or final release.

'You' refers to the licensee, or person using the project, where 'using' is a generic term including compiling the project's source code as well as linking it to form a 'program' or 'executable'. This program is referred to as 'a program using the FreeType engine'.

This license applies to all files distributed in the original FreeType Project, including all source code, binaries and

documentation, unless otherwise stated in the file in its original, unmodified form as distributed in the original archive. If you are unsure whether or not a particular file is covered by this license, you must contact us to verify this.

The FreeType Project is copyright (C) 1996-2000 by David Turner, Robert Wilhelm, and Werner Lemberg. All rights reserved except as specified below.

1. No Warranty

THE FREETYPE PROJECT IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL ANY OF THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DAMAGES CAUSED BY THE USE OR THE INABILITY TO USE, OF THE FREETYPE PROJECT.

2. Redistribution

This license grants a worldwide, royalty-free, perpetual and irrevocable right and license to use, execute, perform, compile, display, copy, create derivative works of, distribute and sublicense the FreeType Project (in both source and object code forms) and derivative works thereof for any purpose; and to authorize others to exercise some or all of the rights granted herein, subject to the following conditions:

- Redistribution of source code must retain this license file ('FTL.TXT') unaltered; any additions, deletions or changes to the original files must be clearly indicated in accompanying documentation. The copyright notices of the unaltered, original files must be preserved in all copies of source files.

- Redistribution in binary form must provide a disclaimer that states that the software is based in part of the work of the FreeType Team, in the distribution documentation. We also encourage you to put an URL to the FreeType web page in your documentation, though this isn't mandatory.

These conditions apply to any software derived from or based on the FreeType Project, not just the unmodified files. If you use our work, you must acknowledge us. However, no fee need be paid to us.

3. Advertising

Neither the FreeType authors and contributors nor you shall use the name of the other for commercial, advertising, or promotional purposes without specific prior written permission.

We suggest, but do not require, that you use one or more of the following phrases to refer to this software in your documentation or advertising materials: 'FreeType Project', 'FreeType Engine', 'FreeType library', or 'FreeType Distribution'.

As you have not signed this license, you are not required to accept it. However, as the FreeType Project is copyrighted material, only this license, or another one contracted with the authors, grants you the right to use, distribute, and modify it. Therefore, by using, distributing, or modifying the FreeType Project, you indicate that you understand and accept all the terms of this license.

4. Contacts

There are two mailing lists related to FreeType:

- freetype@nongnu.org

  Discusses general use and applications of FreeType, as well as future and wanted additions to the library and distribution. If you are looking for support, start in this list if you haven't found anything to help you in the documentation.

- freetype-devel@nongnu.org

  Discusses bugs, as well as engine internals, design issues, specific licenses, porting, etc.

Our home page can be found at

http://www.freetype.org

Component: libogg

Copyright (c) 2002, Xiph.org Foundation

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: libvorbis

Copyright (c) 2002-2004 Xiph.org Foundation

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

- Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

- Neither the name of the Xiph.org Foundation nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Component: zlib

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not
   be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source

```
   distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu

------------------------------------------------------------------------
Component: pcre
------------------------------------------------------------------------

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
   met:

    * Redistributions of source code must retain the above copyright
   notice,
      this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in
   the
      documentation and/or other materials provided with the
   distribution.

    * Neither the name of the University of Cambridge nor the name of
   Google
      Inc. nor the names of their contributors may be used to endorse or
      promote products derived from this software without specific prior
      written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
   IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
   THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
   PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
   THE
POSSIBILITY OF SUCH DAMAGE.

------------------------------------------------------------------------
Component: MeshMagick
------------------------------------------------------------------------

Copyright (c) 2010 Daniel Wickert, Henrik Hinrichs, Sascha Kolewa,
   Steve Streeting

Permission is hereby granted, free of charge, to any person obtaining a
   copy
of this software and associated documentation files (the "Software"),
   to deal
in the Software without restriction, including without limitation the
   rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
```

```
   sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
   in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
   OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
   THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
   FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
   IN
THE SOFTWARE.

-------------------------------------------------------------------------
Component: OgreBullet
-------------------------------------------------------------------------

Copyright 2007 Paul "Tuan Kuranes" Cheyrou-Lagrèze.

This file is part of OgreBullet an integration layer between the OGRE 3D
graphics engine and the Bullet physic library.

Permission is hereby granted, free of charge, to any person obtaining a
   copy
of this software and associated documentation files (the "Software"),
   to deal
in the Software without restriction, including without limitation the
   rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
   sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
   in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
   OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
   THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
   FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
   IN
THE SOFTWARE.

-------------------------------------------------------------------------
Component: OgreProcedural
-------------------------------------------------------------------------
```

```
This source file is part of ogre-procedural

For the latest info, see http://code.google.com/p/ogre-procedural/

Copyright (c) 2010 Michael Broutin

Permission is hereby granted, free of charge, to any person obtaining a
    copy
of this software and associated documentation files (the "Software"),
    to deal
in the Software without restriction, including without limitation the
    rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or
    sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
    in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
    OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
    THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
    FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
    IN
THE SOFTWARE.

-------------------------------------------------------------------
Components:
  - OpenAL
  - OgreAL
  - zziplib
  - Hydrax
-------------------------------------------------------------------

      GNU LIBRARY GENERAL PUBLIC LICENSE
           Version 2, June 1991

 Copyright (C) 1991 Free Software Foundation, Inc.
                    675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the library GPL.  It is
 numbered 2 because it goes with version 2 of the ordinary GPL.]

          Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.
```

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is

analogous to running a utility program or application program.
However, in
a textual and legal sense, the linked executable is a combined work, a
derivative of the original library, and the ordinary General Public
License
treats it as such.

Because of this blurred distinction, using the ordinary General
Public License for libraries did not effectively promote software
sharing, because most developers did not use the libraries. We
concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the
users of those programs of all benefit from the free status of the
libraries themselves. This Library General Public License is intended
to
permit developers of non-free programs to use free libraries, while
preserving your freedom as a user of such programs to change the free
libraries that are incorporated in them. (We have not seen how to
achieve
this as regards changes in header files, but we have achieved it as
regards
changes in the actual functions of the Library.) The hope is that this
will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and
modification follow. Pay close attention to the difference between a
"work based on the library" and a "work that uses the library". The
former contains code derived from the library, while the latter only
works together with the library.

Note that it is possible for a library to be covered by the ordinary
General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which
contains a notice placed by the copyright holder or other authorized
party saying it may be distributed under the terms of this Library
General Public License (also called "this License"). Each licensee is
addressed as "you".

A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work
which has been distributed under these terms. A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language. (Hereinafter, translation is
included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for
making modifications to it. For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation

and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote

it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

  Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

  4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

  5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

  However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

  When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies

the executable.

  It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

  7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

    b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

  11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then

the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if

necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the
> library ‘Frob’ (a library for tweaking knobs) written by James Random
> Hacker.
>
> <signature of Ty Coon>, 1 April 1990
> Ty Coon, President of Vice

That’s all there is to it!

Chapitre 44

Les macros

Syntax

```
Macro <nom> [(Parametre [, ...])]
  ...
EndMacro
```

Description

Le système de macro est une fonctionnalité très puissante, principalement utile pour les programmeurs chevronnés. Une macro est un bout de code quelconque (un mot clef, une ligne, plusieurs lignes) qui peut être inséré à n'importe quel endroit dans le code source en indiquant le nom de la macro. En ce sens, une macro est différente d'une procedure , car les procédures ne dupliquent pas leur code à chaque appel.
La déclaration Macro : EndMacro doit être effectuée avant le premier appel à la macro. Comme les macros seront complètement remplacées par le code correspondant au moment de la compilation, elles ne sont pas locales à une procédure.
Une macro ne peut pas avoir un type de retour, ou des paramètres typés, car cela n'a pas de sens.
Quand une macro a des paramètres, ils sont remplacés dans le code de la macro par l'expression littérale qui a été passée lors de l'appel. Aucune évaluation n'est faite à ce stade, ce qui est très important à comprendre : l'évaluation de la ligne ne commence uniquement lorsque toutes les macros trouvées sur cette ligne ont été traitées.
Les macros sont divisées en deux catégories : les simples (sans paramètres) et les complexes (avec paramètres, obligation d'utiliser les parenthèses pour les appeler). Quand aucun paramètre n'est spécifié, il est possible de remplacer n'importe quel mot avec un autre mot (ou une autre expression). Les macros peuvent être utilisées de manière récursive, mais si un des paramètres passés contient le caractère de concaténation '#', il ne sera pas géré.

Exemple : Macro simple

```
1
2   Macro MonNot
3     Not
4   EndMacro
5
6   a = 0
7   If MonNot a    ; Ici la ligne sera remplacée par : 'If Not a'
8     Debug "Ok"
9   EndIf
```

En utilisant les paramètres, il est possible de créer des macros très flexibles. Le caractère de

concaténation '#' est disponible pour créer des nouveaux mots ou labels en combinant le code de la macro et l'expression passée en paramètre (les espaces ne sont pas acceptées entre chaque mots devant être concaténés). Il est aussi possible de définir des valeurs par défaut pour chaque paramètre, pour qu'ils puissent être omis lors de l'appel de la macro.

Exemple : Macro avec paramètres

```
1  Macro BoiteDeMessageEnMajuscule(Titre, Corps)
2    MessageRequester(Titre, UCase(Corps), 0)
3  EndMacro
4
5  Texte$ = "le Monde"
6  BoiteDeMessageEnMajuscule("Salut", "-"+Texte$+"-") ; Ici la ligne
    sera remplacée comme ça :
7                                                    ;
    MessageRequester("Salut", UCase("-"+Texte$+"-"), 0)
```

Exemple : Macro avec paramètre par défaut

```
1  Macro BoiteDeMessageEnMajuscule(Titre, Corps = "Ah, aucun corps
    spécifié")
2    MessageRequester(Titre, UCase(Corps), 0)
3  EndMacro
4
5  BoiteDeMessageEnMajuscule("Salut") ; Ici la ligne sera remplacée
    comme ça :
6                                     ; MessageRequester("Salut",
    UCase("Ah, aucun corps spécifié"), 0)
```

Exemple : Macro et concaténation

```
1  Macro XCase(Type, Texte) ; renvoie le texte dans la casse du type
    spécifié
2    Type#Case(Texte)         ; Type U => MAJUSCULES
3  EndMacro                  ; Type L => minuscules
4
5  Debug XCase(U, "Salut") ; macro remplacée par UCase("Salut")
6  Debug XCase(L, "Salut") ; macro remplacée par LCase("Salut")
```

Exemple : Macro complexe sur plusieurs lignes

```
1  Macro Guillemet
2    "
3  EndMacro
4
5  Macro Assertion(Expression)
6    CompilerIf #PB_Compiler_Debugger  ; active uniquement l'assertion
    lorsque le débogueur est actif
7      If Expression
8        Debug "Assertion (Ligne " + #PB_Compiler_Line + ") : " +
    Guillemet#Expression#Guillemet
9      EndIf
```

```
    CompilerEndIf
  EndMacro

  Assertion(10 <> 10) ; N'affichera rien
  Assertion(10 <> 15) ; Devrait afficher l'assertion
```

Syntax

```
UndefineMacro <nom>
```

Description

UndefineMacro permet d'annuler une macro définie précédemment, et de la redéfinir d'une manière différente.
Une fois que la macro a été annulée, elle n'est plus disponible.

Exemple : annulation de macro

```
  Macro Test
    Debug "1"
  EndMacro

  Test ; Appel de la macro

  UndefineMacro Test ; Annule la définition du macro, elle n'existe
   plus.

  Macro Test ; Maintenant, nous pouvons redéfinir la macro.
    Debug "2"
  EndMacro

  Test ; Appel de la macro
```

Syntax

```
MacroExpandedCount
```

Description

MacroExpandedCount permet d'obtenir le nombre de fois que la macro a été utilisée/appel. Il peut être utile de générer des identifiants uniques dans la même macro pour chaque utilisation (comme un label, le nom de la procédure, etc.)

Exemple : Comptage

```
  Macro Test
    Debug MacroExpandedCount
  EndMacro

  Test ; Appel de la macro
  Test ; Appel de la macro
  Test ; Appel de la macro
```

Chapitre 45

Pointeurs et accès mémoire

Pointeurs

L'utilisation des pointeurs est possible en plaçant une étoile '*' devant le nom d'une variable d'une liste ou d'une map .
Un pointeur est un emplacement en mémoire qui stocke une adresse mémoire et qui est généralement associé à une structure .

Exemple

```
1  *MonEcran.Ecran = OpenScreen(0,320,200,8,0)
2  mouseX = *MonEcran\SourisX ; La structure Ecran devant contenir un
     champ SourisX
```

Il existe seulement trois méthodes valides pour fixer la valeur d'un pointeur :
- Obtenir le résultat par une fonction (voir l'exemple ci-dessous)
- Copier la valeur d'un autre pointeur
- Trouver l'adresse d'une variable, procédure ou label (voir ci-dessous)
Note : A l'inverse du C/C++, en PureBasic l' '*' fait **partie intégrante** du nom de la variable. Aussi ptr et *ptr sont deux variables bien distinctes.
ptr est une variable (régulière) contenant une valeur, *ptr est une autre variable de type pointeur contenant une adresse.
Pointeurs et taille mémoire
Comme un pointeur reçoit uniquement une adresse mémoire comme valeur, sa taille en mémoire sera celle qui permettra de représenter une adresse du processeur :
- Sur un processeur 32-bit, les adresses sont représentées sur 32-bit, par conséquent un pointeur prendra 32-bit en mémoire (soit 4 octets comme une variable de type long').
- Sur les processeurs 64-bit, les adresses sont représentées sur 64-bit, ce qui implique qu'un pointeur prendra 64-bit en mémoire (soit 8 octets comme une variable de type quad').
C'est pour cette raison qu'un pointeur est une variable dite de type pointeur car son encombrement en mémoire sera lié à la capacité d'adressage mémoire du processeur.
Il en découle qu'affecter un type natif à un pointeur (*Pointeur.l, *Pointeur.b) n'a aucun sens puisque l'encombrement mémoire d'un pointeur est imposé par celui d'une adresse et non par celui d'un type.
Note :
- A chaque fois qu'une adresse mémoire doit être stockée dans une variable, il faudrait le faire par l'intermédiaire d'un pointeur. Ceci garanti que l'adresse sera correctement représentée lors de la compilation du code que ce soit par un processeur 32-bit comme par un processeur 64-bit par exemple.
Pointeurs et structures
Attacher une structure à un pointeur (par exemple *MonPoint.Point) permet d'accéder au contenu mémoire de chaque membre de la structure avec le caractère \ .

Exemple : Pointeurs et variables structurées

```
1  Define Point1.Point, Point2.Point
2  *PointCourant.Point = @Point1 ; Déclare le pointeur, l'associe à une
     structure et l'initialise avec l'adresse de Point1
3  *PointCourant\x = 10          ; Assigne la valeur 10 à Point1\x
4  *PointCourant.Point = @Point2 ; Récupère l'adresse de Point2
5  *PointCourant\x = 20          ; Assigne la valeur 20 à Point2\x
6  Debug Point1\x
7  Debug Point2\x
```

Pointeurs et tableaux
Attacher un tableau à un pointeur permet d'accéder au contenu mémoire de chaque cellules à travers son adresse.

Exemple : Pointeurs et tableaux

```
1  Define Point1.Point, Point2.Point ; 2 Variables de type 'point' (type
     prédéfini dans PureBasic)
2  Dim *Points.Point(1) ; Un tableau de 2 pointeurs et le tableau est de
     type 'Point'
3  *Points(0) = @Point1 ; Le premier pointeur contient l'adresse de la
     variable Point1
4  *Points(1) = @Point2 ; Le second pointeur contient l'adresse de la
     variable Point2
5
6  *Points(0)\x = 10 ; Modification de la variable Point1 à travers le
     pointeur
7  *Points(1)\x = 20 ; Idem avec Point2
8
9  Debug Point1\x
10 Debug Point2\x
```

Les pointeurs permettent donc de se déplacer, de lire et d'écrire facilement en mémoire. De plus ils permettent aux programmeurs d'accéder à de grandes quantités de données sans coût supplémentaire suite à une duplication de ces données. Copier un pointeur est beaucoup plus rapide.
Les pointeurs sont également disponibles dans les structures, pour plus d'informations, consultez le chapitre sur les structures .
Pointeurs et chaînes de caractères
Toutes les variables ont une taille fixe en mémoire (2 octets pour un Word, 4 octets pour un Long, etc) hormis les chaînes de caractères dont la longueur peut changer, ce qui fait qu'elles sont gérées différemment.
Ainsi, les champs d'une structure faisant référence à une chaîne de caractères stockent l'adresse mémoire où réside la chaîne de caractères et non la chaîne elle-même : ce sont des pointeurs vers des chaînes de caractères.

Exemple

```
1  Texte$ = "Bonjour"
2  *Texte = @Texte$ ;*Texte a pour valeur l'adresse où réside la chaîne
     de caractères en mémoire
3  *Pointeur.String = @*Texte ; *Pointeur pointe sur *Texte
4  Debug *Pointeur\s ; Lit la chaîne de caractères qui réside à
     l'adresse écrite en *Pointeur (c-a-d @Texte$)
```

Arithmétiques des pointeurs
Il est possible d'effectuer des opérations arithmétiques sur les pointeurs en s'aidant de la commande SizeOf() .

Exemple

```
1  Dim Tableau.Point(1) ; tableau de points
2
3  *Pointeur.Point = @Tableau() ; Récupère l'adresse du tableau
4  *Pointeur\x = 10 ; Modifie l'élément 0 du tableau
5  *Pointeur\y = 15
6
7  *Pointeur + SizeOf(Point) ; Pointe sur l'élément suivant
8
9  *Pointeur\x = 7 ; Modifie l'élément 1 du tableau
10 *Pointeur\y = 9
11
12 ;Affiche le résultat
13 For i = 0 To 1
14   Debug Tableau(i)\x
15   Debug Tableau(i)\y
16 Next i
```

Adresses des variables : '@'

Pour obtenir l'adresse d'une variable dans votre code, utilisez le symbole @. La raison la plus fréquente d'utiliser ce système est le transfert d'une variable de type structure à une procédure . Il faut passer un pointeur à la procédure car il est impossible de passer directement la structure comme argument.

Exemple

```
1  Structure astruct
2    a.w
3    b.l
4    c.w
5  EndStructure
6
7  Procedure SetB(*monpointeur.astruct)
8    *monpointeur\b = 69
9  EndProcedure
10
11 Define.astruct mavariable
12 mavariable\b = 0
13 SetB( @mavariable )
14 Debug mavariable\b
```

Adresses des chaînes littérales

Pour obtenir l'adresse d'une chaîne de caractères littérales, utilisez le symbole @ devant elle. Les chaînes constantes sont aussi supportées.

Exemple

```
1  *Chaine = @"Test"
2  Debug PeekC(*Chaine) ; Affiche 84, qui est la valeur de la lettre T
     majuscule 'T'
```

Adresses des procédures : '@'

En principe seuls les programmeurs avancés ont à connaître l'adresse d'une procédure. La raison la plus fréquente est de devoir négocier des échanges de bas niveau avec le système d'exploitation. Certains systèmes autorisent la mise en place de callbacks ou points d'ancrage (hooks) permettant au système d'exploitation de dialoguer avec le programme en étendant ainsi les capacités du système d'exploitation. L'adresse d'une procédure est accessible d'une manière similaire à une variable .

Exemple

```
1  Procedure WindowCB(WindowID.i, Message.i, wParam.i, lParam.i)
2    ; C'est ici que le traitement de votre callback sera effectué
3  EndProcedure
4
5  ; Un callback spécifique pour Windows permet de traiter les
     évènements sur les fenêtres
6  SetWindowCallback( @WindowCB() )
```

Adresses des labels : '?'

Il peut également être utile de connaître l'adresse d'un label dans votre programme. Cela peut être le cas pour accéder au code ou aux données placées à cet endroit ou toute autre bonne raison qui peut vous venir à l'esprit. Pour trouver l'adresse d'un label dans votre programme, placez un '?' devant le nom du label.

Exemple

```
1  Debug "Taille du fichier de données = " + Str(?endofmydata - ?mydata)
2
3  DataSection
4    mydata:
5      IncludeBinary "somefile.bin"
6    endofmydata:
```

Chapitre 46

Guide de migration

Introduction

PureBasic est un langage de programmation moderne qui évolue rapidement. Il suit les changements technologiques et apporte une mise à jour de son jeu de fonction, pour le programmeur. Cela implique parfois de modifier ou de réécrire une partie du language. Alors nous essayons de faire ces modifications à minima et ce guide de migration aidera à mettre à jour vos codes sources d'une version vers une autre. Si la stabilité est prioritaire sur les nouveautés, nous vous recommandons fortement de garder la version 'LTS' (Long Term Support/Support à Long Terme) de PureBasic, qui est présentée au public tous les 2 ans, et dont la maintenance active permet la corrections des bugs périodiquement.

Dernière version

migration de 5.50 vers 5.60
migration de 5.30 vers 5.40

Version LTS

migration de 5.20 LTS vers 5.40 LTS

Chapitre 47

Migration de PureBasic 5.20 LTS vers 5.40 LTS

Bibliothèque Billboard

AddBillboard() : code changé

```
; Ancien
AddBillboard(Billboard, BillboardGroup, x, y, z)

; Nouveau
Resultat = AddBillboard(BillboardGroup, x, y, z)
```

Bibliothèque Cipher

ExamineMD5Fingerprint() : code changé

```
; Ancien
ExamineMD5Fingerprint(#FingerPrint)

; Nouveau
UseMD5FingerPrint()
StartFingerprint(#FingerPrint, #PB_Cipher_MD5)
```

ExamineSHA1Fingerprint() : code changé

```
; Ancien
ExamineSHA1Fingerprint(#FingerPrint)

; Nouveau
UseSHA1FingerPrint()
StartFingerprint(#FingerPrint, #PB_Cipher_SHA1)
```

MD5FileFingerprint() : code changé

```
; Ancien
Resultat$ = MD5FileFingerprint(Fichier$)

; Nouveau
UseMD5FingerPrint()
Resultat$ = FileFingerprint(Fichier$, #PB_Cipher_MD5)
```

MD5Fingerprint() : code changé

```
; Ancien
Resultat$ = MD5Fingerprint(*Memoire, Taille)

; Nouveau
UseMD5FingerPrint()
Resultat$ = Fingerprint(*Memoire, Taille, #PB_Cipher_MD5)
```

SHA1FileFingerprint() : code changé

```
; Ancien
Resultat$ = SHA1FileFingerprint(Fichier$)

; Nouveau
UseSHA1FingerPrint()
Resultat$ = FileFingerprint(Fichier$, #PB_Cipher_SHA1)
```

SHA1Fingerprint() : code changé

```
; Ancien
Resultat$ = SHA1Fingerprint(*Memoire, Taille)

; Nouveau
UseSHA1FingerPrint()
Resultat$ = Fingerprint(*Memoire, Taille, #PB_Cipher_SHA1)
```

CRC32FileFingerprint() : code changé

```
; Ancien
Resultat = CRC32FileFingerprint(Fichier$)

; Nouveau
UseCR32FingerPrint()
Resultat.l = Val("$"+FileFingerprint(Fichier$, #PB_Cipher_CRC32))
```

CRC32Fingerprint() : code changé

```
; Ancien
Resultat = CRC32Fingerprint(*Memoire, Taille)

; Nouveau
UseCRC32FingerPrint()
Resultat.l = Val("$"+Fingerprint(*Memoire, Taille, #PB_Cipher_CRC32))
```

NextFingerprint() : renommé

```
; Ancien
NextFingerprint(#FingerPrint, *Memoire, Taille)

; Nouveau
AddFingerprintBuffer(#FingerPrint, *Memoire, Taille)
```

Bibliothèque Mail

SendMail() : code changé si le paramètre 'Asynchronous' est utilisé

```
; Ancien
SendMail(#Mail, Smtp$, Port, 1)

; Nouveau
SendMail(#Mail, Smtp$, Port, #PB_Mail_Asynchronous)
```

Bibliothèque Packer

RemovePackFile() : supprimé
PackerEntrySize() : #PB_Packer_CompressedSize supprimé pour les archives ZIP et 7z

Bibliothèque XML

CreateXMLNode() : code changé

```
; Ancien
Resultat = CreateXMLNode(NoeudParent...)
SetXMLNodeName(Noeud, Nom$)

; Nouveau
Resultat = CreateXMLNode(NoeudParent, Nom$...)
```

Bibliothèque Screen

AvailableScreenMemory() Supprimée car la nouvelle API ne le prend plus en charge. Elle renvoyait toujour un zéro de toute façon.

Bibliothèque Window

#PB_Event_SizeWindow et #PB_Event_MoveWindow ne sont plus en temps réel, utiliser BindEvent() () pour avoir l'action en temps réel.

Engine3D library

WorldCollisionAppliedImpulse() renvoie maintenant un float qui est l'impulsion appliquée.
GetX/Y/Z() ne sont plus supportées.

Divers

- Les étiquettes(labels) en DataSection dans les procédures sont maintenant des étiquettes (labels) locales.
- Les préfixes des étiquettes(labels) locales en ASM ont été changées de "l_" en "ll_", afin d'éviter tout problème avec les étiquettes principales.
- La constante #PB_LinkedList a été renommée en #PB_List pour une meilleure cohérence.

Chapitre 48

Migration de PureBasic 5.30 vers 5.40

Bibliothèque Cipher

ExamineMD5Fingerprint() : code changé

```
; Ancien
ExamineMD5Fingerprint(#FingerPrint)

; Nouveau
UseMD5FingerPrint()
StartFingerprint(#FingerPrint, #PB_Cipher_MD5)
```

ExamineSHA1Fingerprint() : code changé

```
; Ancien
ExamineSHA1Fingerprint(#FingerPrint)

; Nouveau
UseSHA1FingerPrint()
StartFingerprint(#FingerPrint, #PB_Cipher_SHA1)
```

MD5FileFingerprint() : code changé

```
; Ancien
Resultat$ = MD5FileFingerprint(Fichier$)

; Nouveau
UseMD5FingerPrint()
Resultat$ = FileFingerprint(Fichier$, #PB_Cipher_MD5)
```

MD5Fingerprint() : code changé

```
; Ancien
Resultat$ = MD5Fingerprint(*Memoire, Taille)

; Nouveau
UseMD5FingerPrint()
Resultat$ = Fingerprint(*Memoire, Taille, #PB_Cipher_MD5)
```

SHA1FileFingerprint() : code changé

```
; Ancien
```

```
  Resultat$ = SHA1FileFingerprint(Fichier$)

  ; Nouveau
  UseSHA1FingerPrint()
  Resultat$ = FileFingerprint(Fichier$, #PB_Cipher_SHA1)
```

SHA1Fingerprint() : code changé

```
  ; Ancien
  Resultat$ = SHA1Fingerprint(*Memoire, Taille)

  ; Nouveau
  UseSHA1FingerPrint()
  Resultat$ = Fingerprint(*Memoire, Taille, #PB_Cipher_SHA1)
```

CRC32FileFingerprint() : code changé

```
  ; Ancien
  Resultat = CRC32FileFingerprint(Fichier$)

  ; Nouveau
  UseCR32FingerPrint()
  Resultat.l = Val("$"+FileFingerprint(Fichier$, #PB_Cipher_CRC32))
```

CRC32Fingerprint() : code changé

```
  ; Ancien
  Resultat = CRC32Fingerprint(*Memoire, Taille)

  ; Nouveau
  UseCRC32FingerPrint()
  Resultat.l = Val("$"+Fingerprint(*Memoire, Taille, #PB_Cipher_CRC32))
```

NextFingerprint() : renommé

```
  ; Ancien
  NextFingerprint(#FingerPrint, *Memoire, Taille)

  ; Nouveau
  AddFingerprintBuffer(#FingerPrint, *Memoire, Taille)
```

Bibliothèque Mail

SendMail() : code changé si le paramètre 'Asynchronous' a été utilisé

```
  ; Ancien
  SendMail(#Mail, Smtp$, Port, 1)

  ; Nouveau
  SendMail(#Mail, Smtp$, Port, #PB_Mail_Asynchronous)
```

Bibliothèque Packer

RemovePackFile() : supprimé
PackerEntrySize() : #PB_Packer_CompressedSize supprimé pour les archives ZIP et 7z

Bibliothèque Screen

AvailableScreenMemory() Supprimée car la nouvelle API ne le prend plus en charge. Elle renvoyait toujour un zéro de toute façon.

Engine3D library

WorldCollisionAppliedImpulse() renvoie maintenant un float qui est l'impulsion appliquée.
GetX/Y/Z() ne sont plus supportées.

Chapitre 49

Migration de PureBasic 5.50 vers 5.60

Cipher library

Base64Encoder : Fonction renommée

```
1  ; Ancien
2  Base64Encoder()
3
4  ; Nouveau
5  Base64EncoderBuffer()
```

Base64Decoder : Fonction renommée

```
1  ; Ancien
2  Base64Decoder()
3
4  ; Nouveau
5  Base64DecoderBuffer()
```

Autres

La syntaxe autonome 'Define.b', pour changer le type par défaut des variables non typées, est désormais interdite et lèvera une 'Erreur de syntaxe'. Il suffit de supprimer cette instruction de vos code tout simplement (assurez-vous de typer vos variables non typées convenablement).
Remarque : La syntaxe 'Define.b a, b, c' est toujours pris en charge.

Chapitre 50

Module

Syntax

```
DeclareModule <nom>
  ...
EndDeclareModule

Module <nom>
  ...
EndModule

UseModule <nom>
UnuseModule <nom>
```

Description

Les modules sont un moyen facile d'isoler une morceau de code à partir d'un code principal, permettant ainsi la réutilisation d'un code et son partage sans risque de conflit de nom. Dans d'autres langages de programmation, les modules sont connus sous le nom : "espaces de noms". Un module doit avoir une section DeclareModule (qui est l'interface publique) et une section Module associée (qui est son implémentation). Seuls les éléments déclarés dans la section DeclareModule seront accessibles depuis l'extérieur du module. Tout le code de la section Module sera maintenu privé à ce module. Les éléments du code principal, comme les procédures, les variables, etc, ne seront pas accessibles à l'intérieur du module, même s'ils sont déclarés en global . Un module peut être considéré comme une boîte noire, une feuille de code vide où les noms des éléments ne peuvent pas entrer en conflit avec les éléments de même nom du code principal. Il est plus facile d'écrire du code, comme par exemple, on peut utiliser des noms simples qui peuvent être réutilisées au sein de chaque module sans risque de conflit.
Les éléments acceptés dans la section DeclareModule peuvent être les suivants : procédures (seule la déclaration des procédures est autorisée), structures, macros, variables, constantes, énumérations, tableaux, listes, maps et les étiquettes (labels).
Pour accéder à un élément d'un module depuis l'extérieur, le nom du module doit être précisé suivie du séparateur **' : :'**. En spécifiant explicitement le nom du module, l'élément est disponible partout dans le code source, même dans un autre module. Tous les éléments de la section DeclareModule peuvent être automatiquement importés dans un autre module ou dans le code principal à l'aide de UseModule. Dans ce cas, si un nom de module est en conflit, les éléments du module ne seront pas importés et une erreur du compilateur sera levée. UnuseModule retire les éléments du module. UseModule n'est pas obligatoire pour accéder à un élément d'un module, mais le nom du module doit être spécifié.
Pour partager des informations entre les modules, un module commun peut être créé et utilisé par tous les modules qui en ont besoin. C'est la façon normale pour disposer de données globales disponibles pour tous les modules.
Toutes les commandes PureBasic, les structures et les constantes sont des éléments par défaut disponibles dans les modules. Par conséquent les éléments de modules ne peuvent pas avoir le même nom

que les commandes internes de PureBasic, les structures ou les constantes.
Tout les codes à l'intérieur des sections DeclareModule et des sections Module sont exécutés comme n'importe quel autres codes lorsque le flux du programme atteint le module.
Si les mots clés Define, EnableExplicit, EnableASM sont utilisés dans un module, ils n'ont pas d'effet en dehors de ce module.
Quand les états Define, EnableExplicit, EnableASM sont utilisés à l'intérieur d'un module, ils n'ont pas d'effet en dehors du module, ni depuis l'extérieur du module.
Note : Les modules ne sont pas obligatoires dans PureBasic mais sont recommandés lors de la réalisation de grands projets. Ils aident à la maintenance du code, même si c'est légèrement plus verbeux. Avoir une section DeclareModule rend le module plus ou moins auto-documenté pour une réutilisation ou un partage.

Exemple

```
1
2   ; Tous les éléments de cette section seront disponibles de l'extérieur
3   ;
    ------------------------------------------------------------------------------
4   DeclareModule Ferrari
5
6     #FerrariName$ = "458 Italia" ; Constante publique (accessible
    depuis l'extérieur du module)
7     Declare CreateFerrari()      ; La Procédure sera publique
    (accessible depuis l'extérieur du module)
8
9   EndDeclareModule
10
11
12  ; Tous les éléments de cette section seront privés. Tous les noms
     peuvent être utilisés sans conflit
13  ;
    ------------------------------------------------------------------------------
14  Module Ferrari
15
16    Global Initialized = #False
17
18    Procedure Init()             ; Procedure privée Init()
    (inaccessible depuis l'extérieur du module)
19      If Initialized = #False
20        Initialized = #True
21        Debug "InitFerrari()"
22      EndIf
23    EndProcedure
24
25    Procedure CreateFerrari()    ; Procédure publique (car déclarée
    dans la section 'DeclareModule')
26      Init()
27      Debug "CreateFerrari()"
28    EndProcedure
29
30  EndModule
31
32
33  ; Code principal
34  ;
    ------------------------------------------------------------------------------
35  Procedure Init()               ; Procédure d'initialisation
     principale, n'entre pas en conflit avec la procédure
```

```
                                    ; Init() du Module Ferrari

  Debug "Procédure init() du code principal."

EndProcedure

Init()

Ferrari::CreateFerrari()
Debug Ferrari::#FerrariName$

Debug "------------------------------"

UseModule Ferrari                   ; Maintenant, importer tous les
 éléments publics directement dans le programme principal

CreateFerrari()
Debug #FerrariName$
```

Exemple : Avec un module commun

```

; Le module commun, qui sera utilisé par d'autres modules afin de
 partager des données
;
 ------------------------------------------------------------------------------
DeclareModule Voitures
  Global NbVoitures = 0
EndDeclareModule

Module Voitures
EndModule


; Premier module de voiture
;--------------------------
DeclareModule Ferrari
EndDeclareModule

Module Ferrari

  UseModule Voitures
  NbVoitures+1

EndModule


; Second module de voiture
; ------------------------
DeclareModule Porche
EndDeclareModule

Module Porche

  UseModule Voitures
  NbVoitures+1

```

```
35  EndModule
36
37
38  ; Code principal
39  ;
    ------------------------------------------------------------------------------
40
41  Debug Voitures::NbVoitures
```

Chapitre 51

NewList

Syntax

```
NewList nom.<type>()
```

Description

NewList permet de déclarer une nouvelle liste. Chaque élément de la liste est alloué dynamiquement. Il n'y a pas de limite, ce qui permet d'en créer autant que nécessaire. Une liste peut avoir tout type standard ou structuré valide. Pour connaître toutes les commandes disponibles pour la gestion des listes, consultez la bibliothèque List .
Les listes sont toujours locales par défaut, donc pour accéder à partir d'une procédure à une liste définie dans le code source principal du programme, l'utilisation de Global ou Shared est requise. Il est également possible de passer une liste en paramètre d'une procédure à l'aide du mot-clef List.
Utilisez la commande Swap pour permuter rapidement un élément d'une liste avec une variable, ou un élément d'un tableau ou un élément d'une autre liste.

Exemple : Liste simple

```
1   NewList MaListe()
2
3   AddElement(MaListe())
4   MaListe() = 10
5
6   AddElement(MaListe())
7   MaListe() = 20
8
9   AddElement(MaListe())
10  MaListe() = 30
11
12  ForEach MaListe()
13    Debug MaListe()
14  Next
```

Exemple : Liste en paramètre d'une procédure

```
1   NewList Test()
2
3   AddElement(Test())
```

```
  Test() = 1
  AddElement(Test())
  Test() = 2

  Procedure DebugList(c, List ParameterList())

    AddElement(ParameterList())
    ParameterList() = 3

    ForEach ParameterList()
      MessageRequester("Liste", Str(ParameterList()))
    Next

  EndProcedure

  DebugList(10, Test())
```

Chapitre 52

NewMap

Syntax

```
NewMap nom.<type>([Slots])
```

Description

NewMap permet de déclarer une nouvelle map, aussi connue sous le nom de table de hachage ou dictionnaire. Elle permet un accès rapide à un élément basé sur une clef. Chaque clef dans la map est unique, ce qui signifie qu'il n'est pas possible d'avoir deux éléments distincts avec la même clef. Il n'y a pas de limite sur le nombre d'élément que peut contenir une map. Une map peut être déclarée avec n'importe quel type basic ou structuré . Pour consulter toutes les commandes nécessaires à la manipulation des maps, voir la bibliothèque Map .
Quand une nouvelle clef est utilisée lors d'une affectation, un nouvel élément est automatiquement ajouté à la map. Si un autre élément avec la même clef était déjà présent dans la map, il sera remplacé par le nouveau. Une fois qu'un élément a été créé ou accédé, il devient l' élément courant de la map, et les accès à cet élément peuvent ensuite s'effectuer sans avoir à spécifier de clef. C'est très utile lors de l'utilisation d'une map structurée, car la recherche de l'élément ne sera plus nécéssaire pour accéder aux différents champs.
Les maps sont toujours locales par défaut, donc pour accéder à partir d'une procédure à une map définie dans le code source principal du programme, l'utilisation de Global ou Shared est requise. Il est également possible de passer une map en paramètre d'une procédure à l'aide du mot-clef Map.
Utilisez la commande Swap pour permuter rapidement un élément d'une map avec une variable, un élément d'un tableau ou un élément de la map.
Le paramètre optionnel 'Slots' défini le nombre de slots interne qui sera utilisé par la map pour effectuer le stockage des éléments. Plus il y a de slots en interne, plus l'accès à un élément sera rapide, mais plus la consommation mémoire sera importante. C'est un compromis dépendant du nombre d'éléments que la map contiendra au maximum et de la rapidité nécessaire à l'accès d'un élément. La valeur par défaut est 512. Ce paramètre n'a pas d'influence sur le nombre d'éléments que la map peut contenir.

Exemple : Map simple

```
1  NewMap Pays.s()
2
3  Pays("DE") = "Allemagne"
4  Pays("FR") = "France"
5  Pays("UK") = "United Kingdom"
6
7  Debug Pays("FR")
8
9  ForEach Pays()
```

```
    Debug Pays()
  Next
```

Exemple : Map en paramètre d'une procédure

```
NewMap Pays.s()

Pays("DE") = "Allemagne"
Pays("FR") = "France"
Pays("UK") = "United Kingdom"

Procedure DebugMap(Map ParameterMap.s())

  ParameterMap("US") = "United States"

  ForEach ParameterMap()
    Debug ParameterMap()
  Next

EndProcedure

DebugMap(Pays())
```

Exemple : Map structurée

```
Structure Voiture
  Poids.l
  Vitesse.l
  Prix.l
EndStructure

NewMap Voitures.Voiture()

; Ici, nous utilisons l'élément courant après l'insertion du nouvel
 élément
;
Voitures("Ferrari F40")\Poids = 1000
Voitures()\Vitesse = 320
Voitures()\Prix = 500000

Voitures("Lamborghini Gallardo")\Poids = 1200
Voitures()\Vitesse = 340
Voitures()\Prix = 700000

ForEach Voitures()
  Debug "Nom de la Voiture: "+MapKey(Voitures())
  Debug "Poids: "+Str(Voitures()\Poids)
Next
```

Chapitre 53

Autres commandes

Syntax

```
Goto <label>
```

Description

Cette commande permet de transférer directement l'exécution du programme à l'emplacement d'un label. Soyez attentif en utilisant Goto car une mauvaise utilisation peut provoquer une fin anormale du programme...
Note : Pour sortir d'une boucle en toute sécurité, vous devez toujours utiliser Break à la place de Goto et ne l'utilisez pas dans un bloc Select/EndSelect, à moins que vous ayez les aptitudes nécessaire pour gérer la pile vous-même.

Syntax

```
End [CodeDeSortie]
```

Description

Termine et quitte le programme de manière correcte à n'importe quel endroit du code source. Le paramètre optionnel 'CodeDeSortie' peut être utilisé pour renvoyer un code d'erreur différent de 0 (valeur par défaut) lorsque le programme se termine (souvent utilisé par les programmes en mode console).
Le 'CodeDeSortie' peut-être récupéré dans un autre programme à l'aide de la commande ProgramExitCode() .

Syntax

```
Swap <expression>, <expression>
```

Description

Permute la valeur des deux expressions, de manière très rapide. Chaque <expression> doit être soit une variable , un élément (structuré ou non) de tableau , de liste ou de map et être d'un des types natifs : long (.l), quad (.q), string, etc.

Exemple : Permute des chaînes

```
Salut$ = "Salut"
Monde$ = "Monde"

Swap Salut$, Monde$

Debug Salut$+" "+Monde$
```

Exemple : Permute des tableaux à dimensions multiples

```
Dim Tableau1(5,5)
Dim Tableau2(5,5)
Tableau1(2,2) = 10      ; Initialise les tableaux
Tableau2(3,3) = 20

Debug Tableau1(2,2)     ; Affichera 10
Debug Tableau2(3,3)     ; Affichera 20

Swap Tableau1(2,2) , Tableau2(3,3)     ; Permutation de 2 éléments
 entre 2 tableaux différents

Debug "Contenu des tableaux après permutation:"
Debug Tableau1(2,2)     ; Affichera 20
Debug Tableau2(3,3)     ; Affichera 10
```

Exemple : Permute différents éléments

```
Define a, b
Dim Tableau(4, 4)
NewList Liste1.Point()

;-Initialise les variables, le tableau et la liste
a = 11
b = 12

Tableau(1,1) = 21

;Ajoute un élément à la liste 1
AddElement(Liste1())
Liste1()\x = 31
Liste1()\y = 32

;Permute un élément de la liste avec une variable
Swap Liste1()\x, a
;Permute un élément de la liste avec un élément du tableau
Swap Liste1()\y, Tableau(1,1)


;Affiche le résultat
Debug a                 ; Affichera 31
Debug b                 ; Affichera 12
Debug Tableau(1,1)      ; Affichera 32
Debug Liste1()\x        ; Affichera 11
Debug Liste1()\y        ; Affichera 21
```

Chapitre 54

Procedures

Syntax

```
Procedure[.<type>] nom(<variable1[.<type>]> [, <variable2[.<type>]>,
    ...])
  ...
  [ProcedureReturn valeur]
EndProcedure
```

Description

Une procédure est une partie du code indépendante du programme principal. Elle peut avoir des paramètres et ses propres variables . En PureBasic, les procédures sont récursives et peuvent donc s'appeler elles-mêmes. Lors de chaque appel à la procédure, les variables locales sont automatiquement initialisées avec la valeur nulle.
Pour accéder aux variables du programme principal, ils faut les partager au préalable en utilisant les mots clefs Shared ou Global (voir aussi : Protected et Static).
Il est possible d'utiliser des paramètres ayant une valeur par défaut à condition que cette expression soit constante et placée en fin de liste des paramètres. Les paramètres ayant une valeur par défaut pourront être omis lors de l'appel de la procédure, la valeur par défaut de chaque paramètre manquant sera utilisée.
Par exemple : Procedure Calcul(num1, num2=0)
Calcul (2, 3)
Calcul (2)
Une procédure peut également recevoir en paramètre des listes , des maps et des tableaux à l'aide des mot-clefs List, Map et Array
Important : Pour les tableaux vous devrez indiquer le nombre de dimensions.
Par exemple : Procedure Calcul(Array num(3))
Ici num() est un tableau à 3 dimensions !
Une procédure peut renvoyer une valeur de retour. Pour cela, il faut deux choses. Premièrement, il faut définir le type de donnée de retour après Procedure qui peut être 'Integer', 'Long', 'Float', 'String' ou autre puis utiliser le mot clef ProcedureReturn pour déclencher le retour.
Attention ProcedureReturn sort immédiatement d'une procédure, même si l'appel est placé à l'intérieur d'une boucle.
Toutefois ProcedureReturn ne peut pas être utilisé pour renvoyer un tableau , une liste ou une map . Pour cela, passer le tableau, la liste ou la map en tant que paramètre dans la procédure.
Si aucune valeur n'est spécifiée pour ProcedureReturn, la valeur renvoyée sera indéterminée (voir assembleur en ligne pour plus d'information).
Les procédures peuvent également être appelées de manière asynchrone en utilisant les threads .
Les procédures peuvent également contenir une DataSection locale.
Note : Pour déclarer une procedure partagée dans une DLL, voir ProcedureDLL ou ProcedureCDLL .
Pour renvoyer une chaîne de caractères depuis une DLL, voir DLLs .

Note : Pour les programmeurs chevronnés, ProcedureC est disponible pour déclarer la procédure en utilisant la convention d'appel 'cdecl' au lieu de 'stdcall'.

Exemple : procédure qui renvoie une valeur numérique

```
1   Procedure Maximum(nb1, nb2)
2     If nb1>nb2
3       Resultat = nb1
4     Else
5       Resultat = nb2
6     EndIf
7
8     ProcedureReturn Resultat
9   EndProcedure
10
11  Resultat = Maximum(15,30)
12  Debug Resultat
13  End
```

Exemple : Procedure qui renvoie une chaîne de caractères

```
1   Procedure.s Attacher(Texte1$, Texte2$)
2     ProcedureReturn Texte1$ + " " + Texte2$
3   EndProcedure
4
5   Resultat$ = Attacher("Coder avec ","PureBasic"))
6   Debug Resultat$
```

Exemple : Procédure avec un paramètre par défaut

```
1   Procedure a(a, b, c=2)
2     Debug c
3   EndProcedure
4
5   a(10, 12) ; ou
6   a(10, 12, 15)
```

Exemple : Listes en paramètre

```
1   NewList Test.Point()
2
3   AddElement(Test())
4   Test()\x = 1
5   AddElement(Test())
6   Test()\x = 2
7
8   Procedure DebugList(c.l, List ParametreListe.Point())
9
10    AddElement(ParametreListe())
11    ParametreListe()\x = 3
12
13    ForEach ParametreListe()
```

```
      MessageRequester("Liste", Str(ParametreListe()\x))
    Next

  EndProcedure

  DebugList(10, Test())
```

Exemple : Tableau en paramètre

```
  Dim Tableau.Point(10, 15)

  Tableau(0,0)\x = 1
  Tableau(1,0)\x = 2

  Procedure Test(c.l, Array ParametreTableau.Point(2))  ; Attention, ici le tableau comporte 2 dimensions

    ParametreTableau(1, 2)\x = 3
    ParametreTableau(2, 2)\x = 4

  EndProcedure

  Test(10, Tableau())

  MessageRequester("Tableau", Str(Tableau(1, 2)\x))
```

Exemple : ProcedureC

```
  ImportC ""
    qsort(*base, num, taille, *ProcedureComparer)
  EndImport

  Dim valeurs.s(5)
  valeurs(0) = "40"
  valeurs(1) = "10"
  valeurs(2) = "100"
  valeurs(3) = "90"
  valeurs(4) = "20"
  valeurs(5) = "25"

  ProcedureC.i Comparer(*a.String, *b.String)
    ProcedureReturn Val(*a\s) - Val(*b\s)
  EndProcedure

  qsort(@valeurs(), ArraySize(valeurs()) + 1, SizeOf(String), @Comparer())

  For n = 0 To 5
    Debug valeurs(n)
  Next n
```

Syntax

```
Declare[.<type>] nom(<variable1[.<type>]> [, <variable2[.<type>] [=
    ValeurParDefaut]>, ...])
```

Description

Dans certains cas, une procédure peut appeler une autre procédure qui n'a pas été déclarée avant sa propre définition. Ce cas peut se produire et provoquer une erreur de compilation. Declare permet de traiter ce cas particulier en déclarant seulement l'en-tête de la procédure. Il est essentiel que les attributs de la fonction Declare et la déclaration réelle de la procédure soient identiques (type et ValeurParDefaut compris).
Note : Pour déclarer une procedure partagée dans une DLL voir DeclareDLL ou DeclareCDLL .
Note : Pour les programmeurs chevronnés, DeclareC est disponible pour déclarer la procédure en utilisant la convention d'appel 'cdecl' au lieu de 'stdcall'.

Exemple

```
1   Declare Maximum(Valeur1, Valeur2)
2
3   Procedure Traitement()
4     Resultat = Maximum(10, 2)       ; A cet instant Maximum() n'est pas
     connu du compilateur.
5     ProcedureReturn Resultat
6   EndProcedure
7
8   Procedure Maximum(Valeur1, Valeur2)
9     If Valeur1 > Valeur2
10      Resultat = Valeur1
11    Else
12      Resultat = Valeur2
13    EndIf
14
15    ProcedureReturn Resultat
16  EndProcedure
17
18  Debug Traitement()
```

Chapitre 55

Protected

Syntax

```
Protected[.<type>] <variable[.<type>]> [= <expression>] [, ...]
```

Description

Protected permet de créer une variable locale dans une procédure. Elle supplante l'éventuelle variable globale du même nom pendant toute la procédure (contrairement à une variable locale classique non protégée). Une valeur par défaut peut être assignée à la variable. Protected peut aussi être utilisé avec les tableaux , les listes et les maps .
La valeur de la variable locale sera réinitialisée à chaque appel de la procédure. Pour éviter cela, Static permet de déclarer une variable locale indépendante des variables globales tout en gardant sa valeur au fil des appels de la procédure.

Exemple : Avec une variable

```
1  Global a
2  a = 10
3
4  Procedure Change()
5    Protected a
6    a = 20
7  EndProcedure
8
9  Debug a ; Affichera 10 car la variable a été protégée.
```

Exemple : Avec un tableau

```
1  Global Dim Tableau(2)
2  Tableau(0) = 10
3
4  Procedure Change()
5    Protected Dim Tableau(2) ; Ce tableau est protégé, il sera local.
6    Tableau(0) = 20
7  EndProcedure
8
9  Change()
10 Debug Tableau(0) ; Affichera 10 car le tableau a été protégé.
```

Chapitre 56

Prototypes

Syntax

```
Prototype.<type> nom(<parametre>, [, <parametre> [= ValeurDefaut]...])
```

Description

Pour les programmeurs chevronnés. Un Prototype permet la déclaration d'un type particulier qui servira à appeler une fonction. Cela permet de faire facilement des pointeurs de fonctions, car ce type peut être affecté à une variable.
Cette fonctionnalité peut remplacer CallFunction() car elle présente quelques avantages : vérification du type de paramètre, du nombre de paramètres.
Contrairement à CallFunction() , le prototype peut gérer les paramètres de types 'double', 'float' et 'quad' sans aucun problème. GetFunction() permet d'obtenir facilement le pointeur d'une fonction dans une bibliothèque.
Les paramètres en fin de prototype peuvent avoir une valeur par défaut (une expression constante est requise). Les paramètres ayant une valeur par défaut pourront être omis lors de l'appel du prototype, la valeur par défaut de chaque paramètre manquant sera utilisée.
Par défaut, la fonction utilisera la convention d'appel 'stdcall' sur x86, ou 'fastcall' sur x64. Si le pointeur de fonction appelle une fonction C utilisant la convention d'appel 'cdecl', PrototypeC est fortement conseillé.
Les pseudotypes peuvent être utilisés pour les paramètres, mais pas pour le type de retour.

Exemple

```
1  ;MessageRequester("Exemple","Prototype...") ; Décommenter cette ligne
     sous Windows XP
2  Prototype.i ProtoMessageBox(Fenetre.i, Corps$, Titre$, Options.i = 0)
3
4  If OpenLibrary(0, "User32.dll")
5
6    ; 'MsgBox' est une variable de type 'ProtoMessageBox'
7    ;
8    MsgBox.ProtoMessageBox = GetFunction(0, "MessageBoxW")
9
10   MsgBox(0, "Hello", "World") ; Les options peuvent être omises
11 EndIf
```

Exemple : Avec des pseudotypes

```
1  ; Nous spécifions le pseudotype 'p-unicode'pour les 2 paramètres de
     type string
2  ; (texte à afficher et titre) car l'api MessageBoxW est une fonction
     unicode. Le compilateur
3  ; convertira automatiquement les chaînes ascii en unicode pour les
     besoins de la fonction
4  ;
5  ;MessageRequester("Exemple","Prototype...") ; Décommenter cette ligne
     sous Windows XP
6  Prototype.i ProtoMessageBoxW(Fenetre.i, Corps.p-unicode,
     Titre.p-unicode, Options.i = 0)
7
8  If OpenLibrary(0, "User32.dll")
9
10   ; 'MsgBox' est une variable de type 'ProtoMessageBoxW'
11   ;
12   MsgBox.ProtoMessageBoxW = GetFunction(0, "MessageBoxW")
13
14   MsgBox(0, "Hello", "World") ; Les options peuvent être omises
15 EndIf
```

Chapitre 57

Pseudotypes

Description

Pour les programmeurs chevronnés. Les pseudotypes sont destinés à faciliter la programmation quand l'utilisation de bibliothèques externes nécessitant des types non supportés par PureBasic sont requises. Plusieurs types prédéfinis sont disponibles et permettent une conversion à la volée des types PureBasic vers ces types. Etant donné que ce ne sont pas des types normaux, leur notation est volontairement différente : un préfixe 'p-' (pour 'pseudo') fait partie du nom du type. Les pseudotypes disponibles sont :

```
  p-ascii: se comporte comme un type 'string', mais la chaîne de
   caractères sera toujours convertie en
           ascii lors de l'appel de la fonction, même si le programme
   est compilé en mode unicode
.
           C'est très utile pour appeler les fonctions d'une
   bibliothèque qui ne supporte pas
           l'unicode, dans un programme unicode.

  p-utf8: se comporte comme un type 'string', mais la chaîne de
   caractères sera toujours convertie en
          utf-8 lors de l'appel de la fonction. C'est très utile pour
   appeler les fonctions d'une bibliothèque qui ne supporte que
          l'utf-8 comme format de chaîne de caractères.

  p-bstr: se comporte comme un type 'string', mais la chaîne de
   caractères sera toujours convertie en
          bstr lors de l'appel de la fonction. C'est très utile pour
   appeler les fonctions d'une bibliothèque qui ont besoin
          des chaînes de caractères bstr, comme les composants COM.

  p-unicode: se comporte comme un type 'string', mais la chaîne de
   caractères sera toujours convertie en
            unicode lors de l'appel de la fonction, même si le
   programme est compilé en mode ascii.
            C'est très utile pour appeler les fonctions d'une
   bibliothèque qui ne supporte que
            l'unicode, dans un programme ascii.

  p-variant: se comporte comme un type numérique, en ajustant l'appel
   de la fonction pour utiliser correctement
            un paramètre de type 'VARIANT'. C'est très utile pour
   appeler les fonctions d'une bibliothèque qui ont besoin
            des paramètres de type 'VARIANT', comme les composants COM.
```

Les pseudotypes peuvent être utilisés uniquement avec les prototypes , les interfaces et les fonctions importées . Ils ne font la conversion que si c'est nécessaire.

Exemple

```
1  ;MessageRequester("Exemple","Pseudotype...") ; Décommenter cette
     ligne sous Windows XP
2  Import "User32.lib"
3
4    ; Nous spécifions le pseudotype 'p-unicode'pour les 2 paramètres de
     type string
5    ; (texte à afficher et titre) car l'api MessageBoxW est une
     fonction unicode. Le compilateur
6    ; convertira automatiquement les chaînes ascii en unicode pour les
     besoins de la fonction
7    ;
8    MessageBoxW(Window.l, Corps.p-unicode, Titre.p-unicode, Options.i =
     0)
9
10 EndImport
11
12 ; L'exemple suivant fonctionnera correctement bien que le type string
     soit codé en ascii,
13 ; le compilateur se charge de la conversion en unicode pour les
     besoins de la fonction
14 ;
15 MessageBoxW(0, "Hello", "World")
```

Chapitre 58

Les objets PureBasic

Introduction

L'objectif de ce chapitre est d'assimiler la création et la manipulation des objets en PureBasic. Pour cette présentation nous allons utiliser l'objet Image , mais la même logique s'applique à tous les autres objets PureBasic. Quand nous voulons créer une image, nous avons deux possibilités : la méthode indexée et la méthode dynamique.

I. Les objets indexés

La manière indexée (statique) permet de référencer un objet à l'aide d'une valeur numérique que nous déterminons. Elle doit être comprise entre 0 et un nombre maximum qui va dépendre du type de l'objet (en principe entre 5000 et 64000). Ainsi, si vous utilisez la valeur 0 pour votre premier objet et la valeur 1000 pour votre deuxième objet, il y aura 1001 index de disponibles et 999 seront inutilisés, ce qui n'est pas très efficace (gâchis de mémoire vive). C'est pour cette raison qu'il faut autant que possible utiliser une indexation sequentielle, qui commence à 0. Si vous avez besoin d'une méthode plus flexible, il vous faudra probablement utiliser la méthode dynamique, décrite dans la section II. La méthode indexée offre plusieurs avantages :

- Manipulation plus facile, pas besoin de variables ni de tableaux .
- Manipulation 'Groupée' sans avoir à utiliser de tableaux supplémentaires.
- Utilisation des objets dans les procédures sans avoir à déclarer de globales (si on utilise une constante ou un nombre).
- Destruction automatique des objets précédents quand un index est réutilisé.

Les Enumérations sont fortement recommandées si vous souhaitez utiliser des constantes séquentielles pour identifier vos objets.

Exemple

```
1  CreateImage(0, 640, 480) ; Crée une image à l'index n°0 dans
    l'indexation des images
2  ResizeImage(0, 320, 240) ; Redimensionne l'image n°0
```

Exemple

```
1  CreateImage(2, 640, 480) ; Crée une image à l'index n°2 dans
    l'indexation des images
2  ResizeImage(2, 320, 240) ; Redimensionne l'image n°2
3  CreateImage(2, 800, 800) ; Crée une nouvelle image à l'emplacement
    n°2. L'ancienne image n°2 est désallouée automatiquement
```

Exemple

```
1  For k = 0 To 9
2    CreateImage(k, 640, 480) ; Crée 10 images, aux emplacements 0 à 9
3    ResizeImage(k, 320, 240) ; Recrée une nouvelle image à
    l'emplacement n°2. L'ancienne image n°2 est désallouée
    automatiquement
4  Next
```

Exemple

```
1  #ImageBackground = 0
2  #ImageButton     = 1
3
4  CreateImage(#ImageBackground, 640, 480) ; Crée une image à
    l'emplacement #ImageBackground (0)
5  ResizeImage(#ImageBackground, 320, 240) ; Redimensionne l'image 0
    (ImageBackground)
6  CreateImage(#ImageButton    , 800, 800) ; Crée une image (n°1)
```

II. Les objets dynamiques

Quelquefois, les objets indexés ne sont pas pratiques pour gérer des situations où l'on ne connaît pas à l'avance le nombre d'objets nécessaire à un instant donné. Pour cela PureBasic permet de créer très facilement des objets dynamiques. Les deux méthodes (indexée et dynamique) peuvent être utilisées en même temps sans risque de conflit. Pour créer un objet dynamique, il suffit de spécifier la constante #PB_Any à la place du numéro et un numéro dynamique sera retourné comme résultat de la fonction. Cette manière de gérer les objets se marie bien avec les listes , qui sont aussi un moyen dynamique de gérer des données.

Exemple

```
1  DynamicImage1 = CreateImage(#PB_Any, 640, 480) ; Crée une image
    dynamiquement
2  ResizeImage(DynamicImage1, 320, 240) ; Redimensionne l'image
```

Code complet d'example de gestion dynamique d'objets avec une liste chainée :

Présentation des différents objets PureBasic

Différents objets PureBasic (Windows, gadgets, sprites, etc) peuvent utiliser la même énumération de numéros d'objet et pas d'autres. Ainsi, chacun des objets suivants peuvent être énumérés en commençant à 0 (ou autre valeur) car PureBasic les gère par leur type :

- Database
- Dialog
- Entity
- File
- FTP
- Gadget (ScintillaGadget() inclus)
- Gadget3D
- Image
- Library
- Light
- Mail
- Material
- Menu (sauf les MenuItem() qui ne sont pas des objets)
- Mesh
- Movie
- Music
- Network
- Node
- Particle
- RegularExpression
- SerialPort
- Sound
- Sound3D
- Sprite
- StatusBar
- Texture
- ToolBar
- Window
- Window3D
- XML

Chapitre 59

Repeat : Until

Syntax

```
Repeat
  ...
Until <expression> [ou Forever]
```

Description

Cette fonction boucle jusqu'à ce que <expression> soit vrai. Si une boucle infinie est requise il est préférable d'utiliser le mot clef Forever à la place de Until.

Exemple

```
1  a=0
2  Repeat
3    a=a+1
4    Debug a
5  Until a>100
```

Ceci produira une boucle jusqu'à ce que "a" prenne une valeur strictement supérieure à 100, (il y aura donc 101 cycles).

Chapitre 60

Résidents

Description

Les résidents sont des fichiers pré-compilés qui sont chargés lors du démarrage du compilateur. Ils se trouvent dans le dossier des 'residents' dans le chemin d'installation de PureBasic. Un fichier résident doit avoir l'extension '.res' et peut contenir les éléments suivants : Structures , interfaces , macros et constantes . Il ne peut pas contenir un code dynamique ni des procédures .
Lorsqu'un résident est chargé, tout son contenu est disponible pour le programme en cours de compilation. C'est pourquoi toutes les constantes intégrées comme #PB_Event_CloseWindow sont disponibles, elles sont dans le fichier 'PureBasic.res.
Toutes les structures et les constantes de l'API sont également dans un fichier résident. L'utilisation des résidents est un bon moyen pour stocker les macros, les structures et les constantes communes afin qu'elles soient disponibles pour tous les programmes.
Lors de la distribution d'une bibliothèque utilisateur, c'est aussi une bonne solution pour fournir les constantes et les structures nécessaires.
Pour créer un nouveau résident, le compilateur en ligne de commande doit être utilisé, car il n'y a pas d'option pour le faire à partir de l'IDE. Il est souvent nécessaire d'utiliser /IGNORERESIDENT et /CREATERESIDENT en même temps afin d'éviter des erreurs, car la version précédente du résident est chargée avant la création de la nouvelle.
Les résidents aident grandement à avoir une compilation et un démarrage du compilateur plus rapide, car toutes les informations sont stockées au format binaire. C'est beaucoup plus rapide que l'analyse d'un fichier d'inclusion à chaque compilation.

Chapitre 61

Runtime

Syntax

```
Runtime Variable
Runtime #Constante
Runtime declaration Procedure()
Runtime declaration Enumeration
```

Description

Pour les programmeurs chevronnés. Runtime est utilisé pour créer une liste dite ”runtime” (”en cours d'exécution”) pour avoir accès aux objets du programme en cours, comme les variables, les constantes et les procédures. Une fois compilé, un programme n'a plus d'étiquette (labels) ni de variable, ni de constantes ou de noms de procédure car tout est converti en code binaire. Runtime force le compilateur à ajouter une référence supplémentaire à chaque objet qui sera alors disponible à travers la bibliothèque Runtime . Les objets peuvent être manipulés à l'aide de leur référence qui est une chaîne de caractère, même lorsque le programme est compilé.
Pour illustrer l'utilisation d'un Runtime, jetez un coup d'oeil à la bibliothèque Dialog qui l'utilise pour accéder aux procédures d'événement associées à un gadget . Ici, le nom de la procédure à utiliser par le gestionnaire d'événements est spécifié dans le fichier XML (qui est un fichier texte), puis la bibliothèque Dialog utilise la fonction GetRuntimeInteger() pour retrouver l'adresse de la procédure pendant l'exécution du programme. Il n'est pas nécessaire de recompiler le programme pour la changer.
Une autre utilisation serait d'ajouter un petit langage de script en temps réel pour le programme, ce qui permet de modifier facilement les variables exposées, en utilisant les valeurs des constantes à l'exécution. Alors que cela pourrait être fait manuellement par la construction d'une Map d'objets, le mot clé Runtime permet de le faire d'une manière standard et unifiée.

Exemple : Procedure

```
1  Runtime Procedure OnEvent()
2    Debug "OnEvent"
3  EndProcedure
4
5  Debug GetRuntimeInteger("OnEvent()") ; Affichera l'adresse de la
    procédure
```

Exemple : Enumeration

```
Runtime Enumeration
  #Constante1 = 10
  #Constante2
  #Constante3
EndEnumeration

Debug GetRuntimeInteger("#Constante1")
Debug GetRuntimeInteger("#Constante2")
Debug GetRuntimeInteger("#Constante3")
```

Exemple : Variable

```
Define a = 20
Runtime a

Debug GetRuntimeInteger("a")
SetRuntimeInteger("a", 30)

Debug a ; La variable a été modifiée
```

Chapitre 62

Select : EndSelect

Syntax

```
Select <expression1>
  Case <expression> [, <expression> [<expression numerique> To
   <expression numerique>]]
     ...
  [Case <expression>]
     ...
  [Default]
     ...
EndSelect
```

Description

Select permet d'opérer des choix rapides. Le programme exécute <expression1> et retient la valeur en mémoire. Cette valeur est ensuite comparée à chacune des valeurs "Case <expression>" et s'il y a égalité, le code du bloc Case est exécuté pour quitter ensuite la structure Select. Case supporte les valeurs multiples ainsi que les intervalles à l'aide du mot-clef To (seulement pour les intervalles numériques). Si aucune des valeurs Case n'est vraie, alors le code du bloc Default, (s'il est spécifié) est exécuté.
Note : Select accepte les nombres à virgules (float) comme <expression1>, mais ils seront arrondis à l'entier inférieur (les comparaisons ne se font que sur des nombres entiers).

Exemple

```
1   Valeur = 2
2
3   Select Valeur
4     Case 1
5       Debug "Valeur = 1"
6
7     Case 2
8       Debug "Valeur = 2"
9
10    Case 20
11      Debug "Valeur = 20"
12
13    Default
14      Debug "Je ne sais pas"
15  EndSelect
```

Exemple : Cas multiples et intervalles

```
1   Valeur = 2
2
3   Select Valeur
4     Case 1, 2, 3
5       Debug "Valeur est 1, 2 ou 3"
6
7     Case 10 To 20, 30, 40 To 50
8       Debug "Valeur est entre 10 et 20, égale à 30 ou entre 40 et 50"
9
10    Default
11      Debug "Je ne sais pas"
12
13  EndSelect
```

Chapitre 63

Utiliser plusieurs versions de PureBasic avec Windows

Introduction

Il est possible d'installer plusieurs versions de PureBasic sur votre disque dur. C'est utile pour finir un projet avec une ancienne version de PureBasic, tout en commençant le développement d'un nouveau projet avec une version plus récente de PureBasic.

Comment procéder

Créez différents répertoires comme par exemple "PureBasic_v3.94" et "PureBasic_v5" et installez les versions de PureBasic correspondant à chaque répertoire.
Quand un des fichiers "PureBasic.exe" est démarré, il associe tous les fichiers ".pb" à la version de PureBasic correspondante. Aussi quand un code source est chargé en double cliquant sur son fichier, c'est la version de PureBasic associée en cours qui sera démarrée. PureBasic ne changera rien, qui pourrait affecter les autres versions de PureBasic dans les autres répertoires.
Pour éviter l'association automatique des fichiers ".pb" au démarrage de l'IDE, un raccourci peut être créé depuis PureBasic.exe avec "/NOEXT" comme paramètre. La ligne de commande optionnelle depuis l'IDE est décrite ici .
Note : Depuis PureBasic 4.10, les paramètres de l'IDE ne sont plus enregistrés dans le répertoire 'PureBasic' mais dans le répertoire %APPDATA%\PureBasic. Pour utiliser le même fichier de configuration avec différentes versions de Purebasic, utilisez les interrupteurs /P /T et /A. En outre, le commutateur /PORTABLE met tous les fichiers dans le répertoire PureBasic et désactive la création de l'extension '.pb'.

Chapitre 64

Shared

Syntax

```
Shared <variable> [, ...]
```

Description

Shared permet de rendre une variable , un tableau , une liste ou une map non global accessible depuis une procédure . Quand Shared est utilisé avec un tableau, une liste ou une map, seul le nom suivi de '()' doit être spécifié.

Exemple : Avec une variable

```
1   a = 10
2
3   Procedure Change()
4     Shared a
5     a = 20
6   EndProcedure
7
8
9   Change()
10  Debug a   ; Affichera 20, car la variable est partagée.
```

Exemple : Avec un tableau et une liste

```
1   Dim Array(2)
2   NewList List()
3   AddElement(List())
4
5   Procedure Change()
6     Shared Array(), List()
7     Array(0) = 1
8     List() = 2
9   EndProcedure
10
11  Change()
12  Debug Array(0)   ; Affichera 1, car le tableau est partagé.
13  Debug List()     ; Affichera 2, car la liste est partagée.
```

Chapitre 65

Static

Syntax

```
Static[.<type>] <variable[.<type>]> [= <constant expression>] [, ...]
```

Description

Static permet de créer des variables locales persistantes dans une procedure . Les variables statiques sont prioritaires sur les variables globales , ce qui implique qu'une variable globale sera ignorée dans une procédure si une variable statique portant le même nom est déjà déclarée. La valeur de la variable statique n'est pas réinitialisée à chaque appel de la procédure : c'est donc un bon moyen pour avoir une variable globale affectée à une seule procédure.
Static peut aussi être utilisé avec les tableaux , les listes et les maps . Lors de la déclaration d'un tableau static, ses paramètres doivent être une valeur constante.

Exemple : Avec une variable

```
1  Global a
2  a = 10
3
4  Procedure Change()
5    Static a
6    a+1
7    Debug "Dans la Procédure: "+Str(a) ; Affichera 1, 2, 3 car la
     variable s'incrémente à chaque appel de la procédure.
8  EndProcedure
9
10 Change()
11 Change()
12 Change()
13 Debug a ; Affichera 10, car une variable 'static' n'affecte pas une
     variable 'global'.
```

Exemple : Avec un tableau

```
1  Global Dim Tableau(2)
2  Tableau(0) = 10
3
4  Procedure Change()
```

```
5       Static Dim Tableau(2)
6       Tableau(0)+1
7       Debug "Dans la Procédure: "+Str(Tableau(0)) ; Affichera 1, 2, 3 car
        la valeur du champ du tableau s'incrémente à chaque appel de la
        procédure.
8     EndProcedure
9
10    Change()
11    Change()
12    Change()
13    Debug Tableau(0) ; Affichera 10, car un tableau 'static' n'affecte
       pas un tableau 'global'.
```

Exemple : Avec plusieurs procedures

```
1     Procedure Foo()
2       Static x = 100  ; La déclaration et l'affectation sont effectuées
        une seule fois, au lancement du programme.
3
4       Debug x
5       x + 1
6     EndProcedure
7
8     Foo() ; Affiche 100
9     Foo() ; Affiche 101
10    Foo() ; Affiche 102
11
12    Debug "---"
13
14    Procedure Bar()
15      Static x  ; La déclaration est effectuée une seule fois, au
        lancement du programme.
16      x = 100   ; L'affectation est effectuée à chaque lancement de la
        Procedure.
17
18      Debug x
19      x + 1
20    EndProcedure
21
22    Bar() ; Affiche 100
23    Bar() ; Affiche 100
24    Bar() ; Affiche 100
```

Chapitre 66

Structures

Syntax

```
Structure <nom> [Extends <name>] [Align <expression numérique
    constante>]
  ...
EndStructure
```

Description

Structure est utile pour définir un type utilisateur et accéder à des zones mémoires du système d'exploitation par exemple. Les structures peuvent être utilisées pour rendre l'accès à des grands fichiers plus facilement. Cela peut être plus efficace dans la mesure où vous pouvez regrouper dans un même objet des informations communes. On accède aux structures avec le caractère \ . Les structures peuvent s'imbriquer. Les tableaux statiques sont acceptés dans une structure.
Les objets dynamiques tel que les tableaux, listes et maps sont supportés dans les structures et sont automatiquement initialisés quand l'objet utilisant la structure est déclaré. Pour définir un tel champ, utiliser les mot-clés suivant : Array, List et Map.
Le paramètre optionnel Extends permet d'étendre une structure existante avec de nouveaux champs. Tous les champs se trouvant dans la structure étendue se retrouveront en tête de la nouvelle structure. C'est très utile pour faire un héritage simple de structures.
Pour les utilisateurs avancés seulement. Le paramètre Align permet d'ajuster l'alignement entre chaque champ de la structure. L'alignement par défaut est de 1, ce qui signifie pas d'alignement. Par exemple, si l'alignement est fixé à 4, chaque champs sera aligné sur 4 octets. Cela peut aider à améliorer les performances lors de l'accès aux champs de la structure, mais cela peut utiliser plus de mémoire, car un certain espace entre chaque champs sera perdu. La valeur spéciale #PB_Structure_AlignC peut être utilisée pour aligner la structure telle qu'elle se ferait en langage C, utile lors de l'importation structures C utilisées avec des fonctions API.

```
  - SizeOf
permet de connaître la taille en octets d'une structure
  - OffsetOf
peut être utilisé pour rechercher l'index du champ indiqué.
```

Note : Un tableau statique dans une structure ne se comporte pas de la même façon qu'un tableau défini avec la commande Dim . Ceci pour être conforme au format de structures en C/C++ (pour permettre un portage direct des structures de l'API). Ce qui signifie que a[2] assignera un tableau de 0 à 1 (deux éléments) alors que Dim a(2) assignera un tableau de 0 à 2 (trois éléments).
Lorsque vous utilisez des pointeurs dans les structures, L' '*' doit être omis lors de l'utilisation du champ, une fois de plus pour faciliter le portage de code API. Cela peut être considéré comme une bizarrerie (et pour être honnête, ça l'est) mais c'est comme ça depuis le début de PureBasic et beaucoup, beaucoup de sources sont écrites de cette façon et cela restera inchangé.

Quand beaucoup de champs doivent être remplis en une fois, il est conseillé d'utiliser With : EndWith pour réduire la quantité de code à saisir et améliorer sa lisibilité.
Il est possible de copier une structure complète en utilisant l'opérateur égal entre deux éléments de même type.
ClearStructure peut être utilisé pour vider une zone de mémoire structurée. Uniquement pour les programmeurs confirmés, lorsque les pointeurs sont de la partie.

Exemple

```
Structure Personne
  Nom.s
  Prenom.s
  Age.w
EndStructure

Dim MesAmis.Personne(100)

; Ici la position '0' du tableau MesAmis()
; contiendra une personne et ses informations personnelles

MesAmis(0)\Nom = "Durand"
MesAmis(0)\Prenom = "Michel"
MesAmis(0)\Age = 32
```

Exemple : Structure plus complexe (Tableau statique imbriqué)

```
 Structure Fenetre
  *FenetreSuivante.Fenetre  ; Pointe vers un autre objet fenêtre
  x.w
  y.w
  Nom.s[10]  ; 10 noms possibles
EndStructure
```

Exemple : Structure étendue

```
Structure MonPoint
  x.l
  y.l
EndStructure

Structure MonPointEnCouleur Extends MonPoint
  couleur.l
EndStructure

ColoredPoint.MonPointEnCouleur\x = 10
ColoredPoint.MonPointEnCouleur\y = 20
ColoredPoint.MonPointEnCouleur\couleur = RGB(255, 0, 0)
```

Exemple : Copie de structure

```
Structure MonPoint
  x.l
```

```
    y.l
  EndStructure

  PointGauche.MonPoint\x = 10
  PointGauche\y = 20

  PointDroit.MonPoint = PointGauche

  Debug PointDroit\x
  Debug PointDroit\y
```

Exemple : Objet Dynamique

```
  Structure Personne
    Nom$
    Age.l
    List Amis$()
  EndStructure

  Jean.Personne
  Jean\Nom$ = "Jean"
  Jean\Age   = 23

  ; Ajoutons des amis à Jean
  ;
  AddElement(Jean\Amis$())
  Jean\Amis$() = "Jim"

  AddElement(Jean\Amis$())
  Jean\Amis$() = "Monica"

  ForEach Jean\Amis$()
    Debug Jean\Amis$()
  Next
```

Syntax

```
StructureUnion
  Field1.Type
  Field2.Type
  ...
EndStructureUnion
```

Description

StructureUnion est prévu pour les programmeurs avancés qui souhaitent économiser de la mémoire en partageant certains champs à l'intérieur d'une même structure. Il s'agit d'un équivalent du mot clef 'union' en C/C++.
Note : Chaque champ dans la déclaration StructureUnion peut être d'un type différent.

Exemple

```
Structure Type
  Nom$
  StructureUnion
    Long.l      ; Chaque champ (Long, Float et Byte) est placé à la
    Float.f     ; même adresse mémoire.
    String.b    ;
  EndStructureUnion
EndStructure
```

Exemple : Exemple extended (gestion des dates)

```
Structure date
  jour.s{2}
  pk1.s{1}
  mois.s{2}
  pk2.s{1}
  an.s{4}
EndStructure

Structure date2
  StructureUnion
    s.s{10}
    d.date
  EndStructureUnion
EndStructure

Dim d1.date2(5)

d1(0)\s = "05.04.2008"
d1(1)\s = "07.05.2009"

Debug d1(0)\d\jour
Debug d1(0)\d\mois
Debug d1(0)\d\an

Debug d1(1)\d\jour
Debug d1(1)\d\mois
Debug d1(1)\d\an

d2.date2\s = "15.11.2010"

Debug d2\d\jour
Debug d2\d\mois
Debug d2\d\an
```

Exemple : Alignement Mémoire

```
Structure Type Align 4
  Byte.b
  Word.w
  Long.l
  Float.f
EndStructure

```

```
8   Debug OffsetOf(Type\Byte)    ; Affiche 0
9   Debug OffsetOf(Type\Word)    ; Affiche 4
10  Debug OffsetOf(Type\Long)    ; Affiche 8
11  Debug OffsetOf(Type\Float)   ; Affiche 12
```

Exemple : Pointers

```
1   Structure Personne
2     *Next.Personne ; Ici, le '*' est obligatoire pour déclarer un
     pointeur
3     Nom$
4     Age.b
5   EndStructure
6
7   Timo.Personne\Nom$ = "Timo"
8   Timo\Age = 25
9
10  Fred.Personne\Nom$ = "Fred"
11  Fred\Age = 25
12
13  Timo\Next = @Fred ; Lorsque vous utilisez le pointeur, le '*' est omis
14
15  Debug Timo\Next\Nom$ ; Affichera 'Fred'
```

Chapitre 67

Sous-systèmes

Introduction

Les commandes intégrées de PureBasic s'appuient sur les bibliothèques disponibles de chaque système d'exploitation. Parfois, il est possible d'atteindre un même résultat de différentes façons. Pour ce faire, PureBasic offre la possibilité de changer de bibliothèque, sans changer une ligne de code source. Par exemple, sous Windows, OpenGL peut être utiisé en utilisant le sous-système 'OpenGL' de PureBasic, qui lui, utilisera les fonctions OpenGL pour rendre les sprites par exemple, en lieu et place de DirectX (sous-système par défaut).
Pour activer un sous-système, son nom doit être défini dans l'IDE avec le menu Options du compilateur , ou par l'intermédiaire du commutateur /SUBSYSTEM en /ligne de commande . Il s'agit d'une option de compilation, ce qui signifie qu'un exécutable ne peut pas incorporer plus d'un sous-système à la fois. Si le support de multiple sous-systèmes est nécessaire (par exemple l'envoi d'une version OpenGL et DirectX d'un jeu), deux exécutables doivent être créés.
Les sous-systèmes disponibles sont situés dans le dossier 'subsystems' de PureBasic. Lorsqu'un sous-système est spécifié, tous les résidents ou les bibliothèques trouvées dans ce dossier auront préséance sur les bibliothèques par défaut et les résidents. N'importe quel nombre de sous-systèmes différents peuvent être spécifiés (pour autant que cela n'affecte pas les mêmes bibliothèques).
La fonction du compilateur Subsystem peut être utilisée pour détecter si un sous-système spécifique est utilisé pour la compilation.

Sous-systèmes disponibles

Voici une liste des sous-systèmes disponibles, et les bibliothèques concernées :
Windows

```
OpenGL: Utiliser OpenGL au lieu de DirectX. Bibliothèques concernées:
        - Sprite
        - Sprite3D
        - Screen
        - Toutes les bibliothèques connexes du moteur 3D
```

Linux

```
Rien

gtk2: Bibliothèques affectées:

        - 2D Drawing
        - AudioCD
        - Clipboard
        - Desktop
```

- Drag & Drop
- Font
- Gadget
- Image
- Menu
- Movie
- Printer
- Requester
- Scintilla
- StatusBar
- SysTray
- Toolbar
- Window

MacOS X

None

Chapitre 68

Threaded

Syntax

```
Threaded[.<type>] <variable[.<type>]> [= <expression constante>] [, ...]
```

Description

Threaded permet de créer une variable , un tableau sauf les tableaux multi-dimensionnels), une liste ou une map qui sera persistant pour chaque thread . C'est à dire que chaque thread aura sa propre version de l'objet. C'est uniquement utile lors de l'écriture de programmes multi-threadés.
Si un type est spécifié après le mot-clef Threaded, le type par défaut pour cette déclaration est modifié. Chaque variable peut avoir une valeur par défaut assignée, mais cette valeur doit être une constante. Les variables threadées sont initialisées au lancement du premier thread. Cela implique que si la variable est définie est assignée à une valeur en même temps alors elle est définie pour tous les threads. Voir exemple 2. Lors de la déclaration d'un tableau threadé, les paramètres de dimensionnement doivent être des valeurs constantes.
Un objet threadé ne peut pas être déclaré dans une procédure, et sa portée est toujours globale.

Exemple : 1 Avec une variable

```
1  Threaded Compteur
2
3  Compteur = 128
4
5  Procedure Thread(Parametre)
6
7    Debug Compteur ; Affichera zero, car ce thread n'a pas encore
     utilisé cette variable
8    Compteur = 256
9    Debug Compteur ; Affichera 256
10
11 EndProcedure
12
13 Thread = CreateThread(@Thread(), 0)
14 WaitThread(Thread) ; Attente de la fin d'exécution du thread.
15
16 Debug Compteur ; Affichera 128, meme si 'Compteur' a ete change dans
    le thread
```

Exemple : 2 Tous les threads

```
1   Threaded Compteur = 128
2
3   Procedure Thread(Parametre)
4
5     Debug Compteur ; Affichera 128, car quand on lance un programme, on
      lance aussi un thread
6     Compteur = 256
7     Debug Compteur ; Affichera 256
8
9   EndProcedure
10
11  Thread = CreateThread(@Thread(), 0)
12  WaitThread(Thread) ; Attente de la fin d'exécution du thread.
13
14  Debug Compteur ; Affichera 128, meme si 'Compteur' a ete change dans
      le thread
```

Chapitre 69

Unicode

Introduction

'Unicode' est un terme utilisé pour désigner un jeu de caractères étendu destiné à afficher du texte dans de nombreux langages différents (y compris les langages non latins, avec beaucoup de symboles différents tel que le Japonais, le Coréen etc.). En unicode, chaque symbole a sa place propre, et il n'est pas nécessaire d'avoir une table de caractères par langue. Si une application doit fonctionner dans plusieurs langues, il est fortement recommandé d'activer le mode unicode lors du commencement du développement, car cela limitera les bugs qui pourraient survenir en cas de conversion tardive ascii vers unicode.
Pour simplifier, l'unicode peut être vu comme une très grande table ascii, qui n'a pas 256 caractères mais 65536. Donc, pour stocker un caractère, 2 octets seront nécessaires en mémoire (c'est important de noter cette différence, notamment lors de l'utilisation de pointeurs vers des chaînes de caractères). Le type natif 'caractère' (.c) de PureBasic change de taille automatiquement (de 1 octet à 2 octets) si le mode unicode est activé.
Voici quelques liens pour se faire une meilleure idée de l'unicode (lecture vivement conseillée) :
Information générale sur l'unicode (Français)
Information générale sur l'unicode (Anglais)
L'unicode et les BOM

Unicode et Windows

Sous Windows, en mode unicode, PureBasic utilise l'encodage UCS2 qui est le format utilisé de manière interne par l'API, donc aucune conversion n'est nécessaire lorsqu'un fonction est appelée. Si le programme utilise des fonctions API, PureBasic utilisera automatiquement la version unicode de la fonction si elle est disponible (par exemple MessageBox_() pointera vers MessageBoxW() en mode unicode, et vers MessageBoxA() en mode ascii). Il en va de même pour toutes les structures et les constantes API supportées par PureBasic. Ainsi il est possible d'utiliser le même code source API et de le compiler en mode ascii ou unicode sans aucun changement.
Unicode est supporté nativement uniquement sur Windows NT et supérieur (Windows 2000/XP/Vista) : un programme unicode ne fonctionnera pas sur Windows 95/98/NT. Il y a une solution qui consiste à utiliser la dll 'unicows' mais elle n'est pas gérée par PureBasic pour l'instant. Si le programme doit fonctionner sur Win 9x, le mieux est de fournir deux versions de l'exécutable : un en mode ascii et l'autre en mode unicode. Comme il faut uniquement spécifier une option de compilation pour passer du mode ascii au mode unicode, ça devrait être aisé.

UTF-8

L'UTF-8 est une autre façon d'encoder une chaîne de caractères unicode. Contrairement à UCS2 qui prend toujours 2 octets par caractère, UTF-8 utilise un encodage variable pour chaque caractère (jusqu'à 4 octets pour représenter un caractère). Le point fort de l'UTF-8 est qu'il ne contient pas de caractère nul lors de son encodage, donc le texte peut être édité facilement. De plus, les caractères ASCII de 1 à 127 sont conservés (ils ne sont pas modifiés lors de l'encodage) ce qui rend le texte à peu près lisible pour les langues latines. Son point faible est le côté variable de ses caractères, ce qui nécessite des routines de gestion de texte plus lentes.
PureBasic utilise l'UTF-8 comme encodage par défaut pour l'écriture et la lecture des chaînes de caractères dans les fichiers, lorsque l'exécutable est en mode unicode (bibliothèques Fichier et Préférence), pour que les fichiers soient complètement indépendants de la plateforme.
Le compilateur du PureBasic gère aussi les fichiers sources ascii et UTF-8 (les fichiers UTF-8 doivent avoir une entête BOM). Les deux types de fichiers peuvent être combinés dans un programme sans problème : un fichier source ASCII peut inclure un fichier source UTF-8 et vice-versa. Lors du développement d'un programme unicode, il est recommandé de mettre l'IDE en mode UTF-8, pour que tous les fichiers sources soient enregistrés en UTF-8. Comme les sources au format UTF-8 peuvent aussi servir à compiler un programme en mode ASCII, il n'est pas nécessaire de changer le format de fichier à chaque changement de mode.

Chapitre 70

Variables, Types et Opérateurs

Déclaration des variables

La déclaration d'une variable en PureBasic se fait en la nommant. Les variables n'ont pas besoin d'être explicitement déclarées et peuvent être utilisées "à la volée".
Vous pouvez également spécifier le type que vous souhaitez pour cette variable. Cependant, par défaut, une variable qui est déclarée sans indiquer son type de façon explicite sera considérée comme étant de type INTEGER.
Le mot clef Define peut être utilisé pour déclarer plusieurs variables sur une même ligne.

Exemple

```
1  vitesse.q ; Déclare une variable 'vitesse' du type quad (.q).
2  c.l = a*d.w ; 'd' est déclarée ici au milieu d'une expression et elle
     est de type integer (.i) !
3  Define.b a0, b0 = 10, c0 = b0*2, d0
```

Note : Les noms de variables ne peuvent pas commencer par un chiffre (0, 1, etc.), ne peuvent pas contenir de caractères spéciaux(é, à, ß, ä, ö, ü, etc.) ni d'opérateurs (+, -, etc.).
Note : Si le contenu d'une variable ne change pas tout au long de l'exécution du programme (utilisation d'une valeur fixe), il est préférable d'utiliser une constante .

Types basiques

PureBasic permet de définir des variables de plusieurs types comme les entiers, des nombres à virgule, des caractères (char) et des chaînes de caractères aussi.
Voici la liste des types natifs supportés :

```
   Nom          | Extension     | Taille en mémoire                    |
  Plage
--------------+---------------+--------------------------------------+----------------
Byte           |     .b        | 1 octet                              |
  -128 à +127
Ascii          |     .a        | 1 octet                              | 0
  à +255
Caractère      |     .c        | 1 octet (en mode ascii)              | 0
  à +255
Caractère      |     .c        | 2 octets (en mode unicode)           | 0
  à +65535
Word           |     .w        | 2 octets                             |
  -32768 à +32767
```

```
Unicode      |    .u        | 2 octets                              | 0
 à +65535
Long         |    .l        | 4 octets                              |
 -2147483648 à +2147483647
Integer      |    .i        | 4 octets (avec compilateur 32-bit)    |
 -2147483648 à +2147483647
Integer      |    .i        | 8 octets (avec compilateur 64-bit)    |
 -9223372036854775808 à +9223372036854775807
Float        |    .f        | 4 octets                              |
 illimité (voir ci-dessous)
Quad         |    .q        | 8 octets                              |
 -9223372036854775808 à +9223372036854775807
Double       |    .d        | 8 octets                              |
 illimité (voir ci-dessous)
String       |    .s        | longueur string+1                     |
 illimité
String Fixe  | .s{Longueur} | longueur string                       |
 illimité
```

> **Types non-signés (.a, .u et .c) :**
Purebasic offre des types non-signés pour les variables de type 'byte' et 'word' au travers des types 'ascii' (.a) et 'unicode' (.u).
Le type 'character'(.c) est 'word' non-signé en mode unicode .
> **Notation des variables de type chaîne de caractères ($ et .s) :** Généralement une chaîne de caractère se définie avec le type '.s' mais il est possible d'utiliser '**$**' comme dernier caractère d'une variable pour indiquer qu'il s'agit d'une chaîne de caractères. De cette façon vous pouvez utiliser 'a$' et 'a.s' qui sont alors deux variables différentes.
Vous devrez conserver le '$' à la fin de la variable a$ contrairement au '.s' de la variable a.s qui n'est nécessaire que pour sa déclaration.

```
1  a.s = "Une chaîne"
2  a$ = "Une autre chaîne"
3  Debug a   ; Affichera "Une chaîne"
4  Debug a$  ; Affichera "Une autre chaîne"
```

> **Notation scientifique exponentielle (Ae+-B) :** Les nombres à virgules (Float ou Double) peuvent être écrits sous la forme exponentielle :

```
1  valeur.d = 123.5e-20
2  Debug valeur   ; affichera 0.00000000000000001235
```

Opérateurs

Les opérateurs peuvent être intégrés aux expressions pour combiner les variables, constantes et tout ce qui est nécessaire.
La table ci-dessous montre tous les opérateurs utilisables en PureBasic.
LHS = Left Hand Side ou partie gauche de l'équation.
RHS = Right Hand Side ou partie droite de l'équation.

Opérateur = (Egal)

Peut être utilisé suivant deux acceptions. La première est pour l'affectation de la variable LHS à la valeur résultat de l'expression RHS. La seconde signification est l'utilisation dans une expression de comparaison entre LHS et RHS. Si le résultat de LHS est identique au résultat de RHS la valeur 'vrai' sera retournée sinon se sera la valeur 'faux'.

Exemple

```
1  a=b+c ; Affecte la valeur de "b+c" à la variable "a"
2  If abc=def ; Teste si les valeurs de abc et def sont identiques et
    utilise le résultat dans la commande If
```

Opérateur + (Plus)

Donne le résultat de la valeur de l'expression RHS ajoutée à la valeur de l'expression LHS. Si le résultat de cet opérateur n'est pas utilisé et qu'il y a une variable LHS, alors la valeur de l'expression RHS sera directement ajoutée à la valeur LHS.

Exemple

```
1  nombre=mavaleur+2 ; Ajoute la valeur 2 à "mavaleur" et utilise le
    résultat avec l'opérateur =
2  variable+expression ; La valeur de "expression" est directement
    ajoutée à "variable"
```

Opérateur - (Moins)

Soustrait la valeur de l'expression RHS de la valeur de l'expression LHS. S'il n'y a pas d'expression LHS l'opérateur prend la valeur négative de la valeur RHS. Si le résultat de l'opérateur n'est pas utilisé et qu'il n'y a pas de variable LHS, alors la valeur de RHS est directement soustraite à la valeur de la variable LHS. Cet opérateur ne peut être utilisé avec les variables de type chaîne.

Exemple

```
1  var=#MaConstante-chose; Soustrait la valeur de "chose" de
    "#MyConstant" et utilise le résultat avec l'opérateur égal.
2  uneautre=uneautre+ -var ; Calcule la valeur négative de "var" et
    utilise le résultat avec l'opérateur +.
3  variable-expression ; La valeur "expression" est directement
    soustraite à "variable"
```

Opérateur * (Multiplication)

Multiplie la valeur de l'expression LHS par la valeur de RHS. Si le résultat de l'opérateur n'est pas utilisé et qu'il y a une variable LHS, alors la valeur de la variable est directement multipliée par la valeur de l'expression RHS. Cet opérateur ne peut être utilisé dans une variable de type chaine.

Exemple

```
1  total=prix*quantite ; Multiplie la valeur de "prix" par la valeur de
    "quantite" et utilise le résultat avec l'opérateur =
2  variable*expression ; "variable" est multiplé directement par la
    valeur de "expression"
```

Opérateur / (Division)

Divise la valeur de l'expression LHS par la valeur de l'expression RHS. Si le résultat de l'opérateur n'est pas utilisé et qu'il y a une variable LHS, alors la valeur de la variable est directement divisée par la valeur de l'expression RHS. Cet opérateur ne peut être utilisé dans les variables de type chaine.

Exemple

```
1  quantite=total/prix ; Divise la valeur "total" par la valeur "prix"
     et utilise le résultat avec l'opérateur =
2  variable/expression ; "variable" est directement divisé par la valeur
     "expression"
```

Opérateur & (AND est un ET logique (binaire))

Il vous faut être familiarisé avec les nombres binaires pour utiliser cet opérateur. Le résultat de cet opérateur est le résultat d'un ET logique entre les valeurs des expressions LHS et RHS, bit à bit. La valeur de chaque bit résultant est fixée comme indiqué dans la table ci-dessous. De plus, si le résultat de l'opérateur n'est pas utilisé et qu'il y a une variable LHS, alors le résultat sera directement stocké dans cette variable. Cet opérateur ne peut être utilisé avec une variable de type chaîne.

```
LHS | RHS |Résultat
------------------
 0  |  0  |    0
 0  |  1  |    0
 1  |  0  |    0
 1  |  1  |    1
```

Exemple

```
1  ; La représentation binaire des valeurs est utilisée pour une
     présentation plus claire et lisible
2  a.w = %1000 & %0101 ; Le résultat sera 0
3  b.w = %1100 & %1010 ; Le résutlat sera %1000
4  bits = a & b ; Effectue un ET bit à bit entre a et b et utilise le
     résultat avec l'opérateur =
5  a & b ; Effectue un ET bit à bit entre a et b et place le résultat
     directement dans la variable "a"
```

Opérateur || (OR est un OU logique (binaire))

Vous devez être familiarisé avec les nombres binaires pour utiliser cet opérateur. Le résultat de cet opérateur est le résultat d'un OU logique entre les valeurs des expressions LHS et RHS bit à bit. La valeur de chaque bit résultant est fixée comme indiqué dans la table ci-dessous. De plus, si le résultat de l'opérateur n'est pas utilisé et qu'il et qu'il y a une variable LHS alors le résultat est directement stocké dans cette variable. Cet opérateur ne peut être utilisé avec une variable de type chaîne.

```
LHS | RHS |Résultat
------------------
 0  |  0  |    0
 0  |  1  |    1
 1  |  0  |    1
 1  |  1  |    1
```

Exemple

```
1  ; La représentation binaire des valeurs est utilisée pour une
     présentation claire et lisible
2  a.w = %1000 | %0101 ; Le résultat sera %1101
3  b.w = %1100 | %1010 ; Le résultat sera %1110
4  bits = a | b ; Effectue un OU bit à bit entre a et b et utilise le
     résultat avec l'opérateur =
5  a | b ; Effectue un OU bit à bit entre a et b et place le résultat
     directement dans la variable "a"
```

Opérateur ! (XOR est un OU exclusif logique (binaire))

Vous devez être familiarisé avec les nombres binaires pour utiliser cet opérateur. Le résultat de cet opérateur est le résultat d'un OU Exclusif entre les valeurs LHS et RHS bit à bit. La valeur de chaque bit résultant est fixée comme indiqué dans la table ci-dessous. De plus, si le résultat de l'opérateur n'est pas utilisé et qu'il y a une variable LHS alors le résultat est directement stocké dans cette variable. Cet opérateur ne peut être utilisé avec une variable de type chaîne.

```
LHS | RHS |Résultat
------------------
 0  |  0  |    0
 0  |  1  |    1
 1  |  0  |    1
 1  |  1  |    0
```

Exemple

```
1  ; La représentation binaire des valeurs est utilisée pour une
     présentation claire et lisible
2  a.w = %1000 ! %0101 ; Le résultat sera %1101
3  b.w = %1100 ! %1010 ; Le résultat sera %0110
4  bits = a ! b ; Effectue un OU Exclusif bit à bit entre a et b et
     utilise le résultat avec l'opérateur =
5  a ! b ; Effectue un OU Exclusif bit à bit entre a et b et place le
     résultat directement dans "a"
```

Opérateur * * (NON inversion logique (binaire))

Vous devez être familiarisé avec les nombres binaires pour utiliser cet opérateur. Le résultat de cet opérateur est une inversion bit à bit de la valeur RHS. La valeur de chaque bit est fixée comme indiqué dans la table ci-dessous. Cet opérateur ne peut être utilisé avec une variable de type chaîne.

```
RHS |Résultat
----------
 0  |    1
 1  |    0
```

Exemple

```
1  ; La représentation binaire des valeurs est utilisée pour une
     présentation claire et lisible
```

```
2   a.w = ~%1000 ; Le résultat sera %0111
3   b.w = ~%1010 ; Le résultat sera %0101
```

() (Parenthèses)

Vous pouvez utiliser les parenthèses pour forcer l'évaluation prioritaire d'une partie d'une expression ou modifier l'ordre d'évaluation.

Exemple

```
1   a = (5 + 6) * 3 ; Le résultat est 33 car 5+6 est évalué en premier
2   b = 4 * (2 - (3 - 4)) ; Le résultat est 12 car 3-4 est évalué en
      premier, ensuite 2-résultat puis la mutiplication pour finir
```

< (Inférieur à)

Utilisé pour comparer les valeurs des expressions LHS et RHS. Si la valeur de LHS est plus petite que la valeur de RHS cet opérateur rend un résultat vrai, sinon le résultat est faux.

> (Supérieur à)

Utilisé pour comparer les valeurs des expressions LHS et RHS. Si la valeur de LHS est plus grande que la valeur de RHS cet opérateur rend un résultat vrai, sinon le résultat est faux.

<=, =< (Inférieur ou égal à)

Utilisé pour comparer les valeurs des expressions LHS et RHS. Si la valeur de LHS est plus petite ou égale à la valeur de RHS cet opérateur rend un résultat vrai, sinon le résultat est faux.

>=, => (Supérieur ou égal à)

Utilisé pour comparer les valeurs des expressions LHS et RHS. Si la valeur de LHS est plus grande ou égale à la valeur de RHS cet opérateur rend un résultat vrai, sinon le résultat est faux.

<> (Différent)

Utilisé pour comparer les valeurs des expressions LHS et RHS. Si la valeur de LHS est différente de la valeur de RHS cet opérateur rend un résultat vrai, sinon le résultat est faux.

And (ET logique)

Peut être utilisé pour combiner les résultats vrais ou faux des opérateurs de comparaison en donnant un résultat fixé comme indiqué dans la table ci-dessous.

```
  LHS   |   RHS   |Résultat
------------------------
  faux  |   faux  |  faux
  faux  |   vrai  |  faux
  vrai  |   faux  |  faux
  vrai  |   vrai  |  vrai
```

Or (OU logique)

Peut être utilisé pour combiner les résultats vrais ou faux des opérateurs de comparaison en donnant un résultat fixé comme indiqué dans la table ci-dessous.

```
  LHS   |   RHS   | Résultat
-------------------------
  faux  |   faux  |  faux
  faux  |   vrai  |  vrai
  vrai  |   faux  |  vrai
  vrai  |   vrai  |  vrai
```

Operator XOr (OU exclusif logique)

Peut être utilisé pour combiner les résultats vrais ou faux des opérateurs de comparaison en donnant un résultat fixé comme indiqué dans la table ci-dessous.

```
  LHS  |   RHS  | Résultat
------------------------
 faux  | faux   | faux
 faux  | vrai   | vrai
 vrai  | faux   | vrai
 vrai  | vrai   | faux
```

Operator Not (NON logique)

Le résultat de cet opérateur sera la négation de l'expression RHS. Cet opérateur ne fonctionne pas avec les strings.

```
  RHS  | Résultat
----------------
 faux  | vrai
 vrai  | faux
```

Opérateur « (Décalage à gauche)

Décale vers la gauche les bits du nombre LHS de RHS places. Décaler les bits vers la gauche revient à faire une multiplication par un multiple de 2. Il est conseillé de bien comprendre les opérations binaires avant d'utiliser cet opérateur.

Exemple

```
1  a=%1011 << 1 ; La valeur de 'a' sera %10110. (en decimal: %1011=11 et
     %10110=22)
```

```
2  b=%111 << 4 ; La valeur de 'b' sera %1110000. (en decimal: %111=7 et
     %1110000=112)
3  c.l=$80000000 << 1 ; La valeur de 'c' sera 0. Les bits supérieurs
     sont perdus car ils dépassent la capacité du type.
```

Opérateur » (Décalage à droite)

Décale vers la droite les bits du nombre LHS de RHS places. Décaler les bits vers la droite revient à faire une division par un multiple de 2. Il est conseillé de bien comprendre les opérations binaires avant d'utiliser cet opérateur.

Exemple

```
1  d=16 >> 1 ; La valeur de 'd' sera 8. (en binaire: 16=%10000 et
     8=%1000)
2  e.w=%10101010 >> 4 ; La valeur de 'e' sera %1010. (en décimal:
     %10101010=170 et %1010=10).
3  f.b=-128 >> 1 ; La valeur de 'f' sera -64. -128=%10000000,
     -64=%11000000. Lors du décalage, le bit le plus fort reste
     (conservation du signe).
```

Opérateur % (Modulo)

Calcule le reste de la division entière de RHS par LHS.

Exemple

```
1  a=16 % 2 ; La valeur sera 0 car 16/2 = 8 (aucun reste)
2  b=17 % 2 ; La valeur sera 1 car 17/2 = 8*2+1 (reste 1)
```

Opérateurs raccourcis

Tous les opérateurs mathématiques peuvent être utilisés sous une forme abrégée.

Exemple

```
1  Valeur + 1   ; Equivaut à : Valeur = Valeur + 1
2  Valeur * 2   ; Equivaut à : Valeur = Valeur * 2
3  Valeur << 1  ; Equivaut à : Valeur = Valeur << 1
```

Note : Cela peut conduire à des résultats "imprévus" dans quelques rares cas, si l'assignement est modifiée avant l'affection.

Exemple

```
1  Dim MonTableau(10)
2  MonTableau(Random(10)) + 1 ; Equivaut à: MonTableau(Random(10)) =
    MonTableau(Random(10)) + 1, mais ici Random() ne renverra pas la
    même valeur à chaque appel.
```

Priorité des opérateurs

```
Niveau de Priorité |     Opérateurs
-------------------+----------------------------
     8 (haut)      |      ~, - (ici - est le négatif)
     7             |    <<, >>, %, !
     6             |       |, &
     5             |       *, /
     4             |       +, - (ici - est la soustraction)
     3             | >, >=, =>, <, <=, =<, =, <>
     2             |       Not
     1 (bas)       |  And, Or, XOr
```

Types structurés

Les types structurés peuvent être définis avec les options propres aux structures. Voyez le chapitre structures pour plus d'informations.

Types Pointeur

Les pointeurs sont déclarés avec un '*' devant le nom de la variable. Plus d'informations peuvent être trouvées dans le chapitre pointeurs .

Informations concernant les nombres flottants

Un nombre flottant est stocké de telle manière que la 'virgule flotte' autour de la partie réelle. De la sorte, il est possible d'avoir des nombres dont la valeur peut être aussi bien grande que petite. Toutefois vous ne pouvez pas stocker de grands nombres avec une précision aussi élevée que des petits nombres. Une autre limitation concernant les nombres flottants est qu'ils restent concrètement représentés sous une forme binaire. Ainsi, ils ne peuvent être restitués qu'à partir de multiples et de divisions en base 2. Celà est important pour comprendre que la représentation décimale lors de l'affichage ou du calcul n'est pas tout à fait identique à ce que l'on peut attendre dans une représentation humaine. Représenter 0.5 ou 0.125 est simple car ce sont des divisions parfaites de 2, cela est plus complexe pour des nombres comme 0.11 ou 0.10999999. L'affichage approché de la valeur est toujours correct à un nombre limité de décimales, mais ne soyez pas surpris si au-delà le nombre affiché s'écarte de la valeur que vous attendez ! Ces remarques s'appliquent aux nombres flottants traités par ordinateur d'une manière générale et non spécifiquement à Purebasic.
Comme leur nom l'indique, les 'doubles' sont des flottants 'double-precision' (64-bit) comparativement aux flottants 'simple-precision' que sont les floats (32-bit). Donc, pour avoir plus de précision dans la manipulation des nombres à virgule, il est préférable d'utiliser les 'doubles'.

Float : De +- 1.175494e-38 à +- 3.402823e+38
Double : De +-; 2.2250738585072013e-308 à +- 1.7976931348623157e+308

Pour plus d'information sur le format 'IEEE 754', consulter l'article Wikipedia.

Exemple

```
1     a.f=0.1
2     b.f=0.5
3     c.f=0.9
4     Debug a ; Affiche 0.10000000149012
5     Debug b ; Affiche 0.5
6     Debug c ; Affiche 0.89999997615814
```

Chapitre 71

AddPathSegment Suite

Traduction FR de la page Standard SVG Tiny

Voir aussi

PathSegments() , AddPathSegments()

Chapitre 72

While : Wend

Syntax

```
While <expression>
  ...
Wend
```

Description

While produit une boucle jusqu'à ce que l'expression devienne fausse. Il est à noter qu'avec le test accompagnant While, si la première tentative échoue, le programme n'entrera pas dans la boucle et évitera donc cette section de code contrairement à une boucle Repeat qui est toujours exécutée au moins une fois (car le test est effectué en sortie de code conditionnel).

Exemple

```
1  b = 0
2  a = 10
3  While a = 10
4    b = b+1
5    If b=10
6      a=11
7    EndIf
8  Wend
```

Ce programme boucle jusqu'à ce que la valeur "a" devienne différente de 10. La boucle sera donc exécutée 10 fois.

Chapitre 73

Gestion des messages Windows

Introduction

Les messages de votre programme seront transférés par Windows dans une file d'attente, qui est traitée uniquement si vous le désirez. Windows transmet un millier de messages à votre programme sans avertissement direct.
Par exemple si vous changez le statut d'un gadget (que ce soit en ajoutant une saisie ou en changeant l'image d'un ImageGadget), un message est envoyé à la file d'attente de votre programme.
Il y a deux possibilités de récupérer et de traiter les messages Windows dans PureBasic : WaitWindowEvent() and WindowEvent() . La différence est, que WaitWindowEvent() attend qu'un message arrive alors que WindowEvent() permet de continuer à travailler. Les messages dans la file d'attente ne seront cependant traités qu'après l'appel à WindowEvent() ou WaitWindowEvent().

Spécificités de WindowEvent()

La commande WindowEvent() n'attend pas, qu'un message arrive, mais vérifie seulement s'il y en a un dans la file d'attente. Si Oui, le message est traité et WindowEvent() renvoie le numéro du message. Si aucun message n'est dans la file, alors zéro (0) est renvoyé.

Exemple

```
1  While WindowEvent() : Wend
```

Fait en sorte, que WindowEvent() soit appelée tant qu'elle ne renvoie pas 0, c'est à dire jusqu'à ce que tous les messages de la file d'attente soient traités
Insérer un simple 'WindowEvent()' après un SetGadgetState() ne suffit pas pour traiter ce message en particulier. Premièrement il peut toujours y avoir d'autres messages dans la file d'attente, qui sont arrivés auparavant, et deuxièmement Windows envoie également un nombre conséquent d'autres messages, dont nous n'avons que faire... mais qui néanmoins sont dans la file d'attente.
Un simple appel à

```
1  WindowEvent()
```

ne suffit pas, le programme peut fonctionner correctement dans certaines circonstances sur une version de Windows, mais pas sur une autre version. Les différentes versions de Windows ont un fonctionnement interne tellement spécifique, qu'une version envoie seulement 1 message mais une autre version peut envoyer 5 messages pour le même cas de figure. .
A cause de cela, on utilise toujours pour la mise à jour :

```
1   While WindowEvent() : Wend
```

Bien sûr il y a aussi l'alternative

```
1   Repeat : Until WindowEvent() = 0
```

possible, qui n'est cependant pas très courante.
La méthode While WindowEvent() : Wend est fréquemment utilisée avec la commande Delay() , lorsqu'une une boucle est insérée AVANT le Delay(), par exemple pour attendre que la mise à jour d'un ImageGadget soit effective, après avoir changé une image avec SetGadgetState().

Chapitre 74

With : EndWith

Syntax

```
With <expression>
  ...
EndWith
```

Description

Le bloc With : EndWith permet de réduire la quantité de code à saisir et améliore sa lisibilité quand beaucoup de champs d'une structure doivent être renseignés. L'<expression> spécifiée sera automatiquement insérée avant chaque caractère anti-slash '\' qui suit un espace ou un opérateur. C'est une directive de compilateur qui fonctionne de la même manière qu'une macro : la ligne complète est recréée lors de la compilation, puis elle est traitée. Les blocs With : EndWith ne peuvent pas être imbriqués, car cela pourrait générer des bugs difficiles à résoudre.

Exemple

```
1   Structure Personne
2     Nom$
3     Age.l
4     Taille.l
5   EndStructure
6
7   Ami.Personne
8
9   With Ami
10    \Nom$ = "Yann"
11    \Age   = 30
12    \Taille  = 196
13
14    Debug \Taille+\Taille
15  EndWith
```

Exemple

```
1   Structure Corps
2     Poids.l
3     Couleur.l
```

```
    Texture.l
  EndStructure

  Structure Personne
    Nom$
    Age.l
    Corps.Corps[10]
  EndStructure

  Ami.Personne

  For k = 0 To 9
    With Ami\Corps[k]
      \Poids = 50
      \Couleur  = 30
      \Texture = \Couleur*k

      Debug \Texture
    EndWith
  Next
```