Jeremy Gordon

avec la collaboration de *Wayne J. Radburn*

Assembleur 32/64 bits

GoLink Éditeur de liens

GoRC Compilateur de ressources

GoBug Débogueur symbolique

Unicode

A "Go" development tool: http://www.GoDevTool.com

Contacts :

Jeremy Gordon – info@goprog.com

GoAsm Assembler and Tools forum

(dans le Forum MASM)

Volume 2

Traduction française

Robert Cordonnier – Orléans

Version 1.0.3 – 24/06/2024 à 11h01

robert.cordonnier@wanadoo.fr

Sommaire

Chapitre 1 – Éditeur de liens GoLink

1.1 Pourquoi un nouvel éditeur de liens ?

- **Simplicité** : J'ai réduit le nombre de fichiers au minimum et supprimé les fichiers LIB. Le linker est maintenant gratuit sans problème pour ce qu'il est censé faire, à savoir lier un ou plusieurs fichiers objet et créer l'exécutable final.
- **Vitesse** : GoLink est, bien sûr, écrit en assembleur. Pour cette raison, et aussi parce qu'il peut vérifier les détails des importations à partir de DLL qui sont, pour la plupart, déjà en mémoire, il est très rapide.
- **Indépendance** : les programmeurs Win32 qui utilisent l'assembleur ont des contraintes de syntaxe pour rendre leurs fichiers compatibles avec les outils Microsoft. C'est fini. GoLink fait partie de la suite d'outils GoAsm, GoRC et GoBug qui sont finement ajustés pour travailler ensemble. Dans ce contexte, ils sont capables de créer des programmes Win32 en utilisant l'assembleur très simplement.

1.2 Les fonctionnalités de GoLink en bref

- GoLink procure une sortie précise. Ceux qui aiment analyser l'intérieur des exécutables peuvent apprécier la sortie de GoLink qui maintient les "bagages" au minimum. Le visualiseur PEview PE / COFF de Wayne Radburn, disponible gratuitement depuis ici, permet de contrôler cette qualité.
- GoLink combine les fichiers d'objet COFF (Common Object File Format) PE (Portable Executable) et crée des exécutables PE appropriés pour fonctionner sur le système d'exploitation Windows. Il peut créer des fichiers EXE, DLL ou SYS.
- GoLink peut également lier un fichier contenant des ressources au format de fichier RES ou OBJ.
- GoLink peut produire des exécutables 32 bits pour les processeurs 386 et plus récents (x86) ou des exécutables 64 bits pour la famille de processeurs AMD64 ou Intel EM64T (x86-64). La commutation entre les deux est automatique : aucun commutateur de ligne de commande n'est requis.
- GoLink fournit, au besoin, une sortie de débogage sous forme de symboles COFF ou COFF intégrés dans un fichier séparé. Cela vous permet d'utiliser un débogueur symbolique (tel que GoBug) sur votre exécutable.
- Caractéristique majeure, GoLink n'a pas besoin de fichiers LIB (bibliothèque). A la place, il va chercher à l'intérieur-même des exécutables d'export les fonctions et les données importées. Cela rend GoLink beaucoup plus facile à utiliser que d'autres linkers et accélère également le processus d'édition de liens.
- GoLink est finement ajusté sur GoAsm et, utilisés conjointement, ces deux outils balaient les exigences pesantes requises précédemment pour l'import et l'export de fonctions et de données entre un exécutable principal et ses DLL.
- Dans la mesure où il utilise aussi les fichiers objet issus de GoAsm (ou les fichiers compatibles Go), GoLink peut établir un diagnostic des données et symboles de code non utilisés et non référencés. Cela vous permet d'éliminer facilement les déclarations de données et de code redondantes dans vos programmes. Voir le commutateur /unused.
- En utilisant le commutateur /mix, GoLink ignore tout symbole d'enrichissement qui pourrait être inséré par d'autres compilateurs et assembleurs. Il en découle que vous pouvez lier des objets à partir de ces autres outils. Voir le commutateur /mix.
- GoLink peut lire une ligne de commande libellée en Unicode et des fichiers de commande également en Unicode, de sorte que vous pouvez utiliser les noms de fichiers Unicode dans vos projets. Il traite également correctement l'Unicode et des labels de données qu'il pourrait trouver dans les fichiers objet.

1.3 Utilisation de GoLink

Vous pouvez lier un fichier très simplement à partir d'une fenêtre d'invite de commande MS-DOS, par exemple :

```
GoLink MyFile.obj
```

permet de créer MyFile.exe à partir du fichier MyFile.obj.

1.3.1 Fichiers de commande

Dans la pratique, GoLink réclame beaucoup plus d'informations sur les fichiers à traiter et utilise également les commutateurs de ligne de commande. Pour plus de commodité, ces spécifications peuvent être regoupées dans un ou plusieurs *fichiers de commande* placés à leur tour dans la ligne de commande comme suit :

```
GoLink @command.fil              ; exemple avec un fichier de commande
```

```
GoLink @hello @goodbye.def       ; exemple avec deux fichier de commande
```

Un fichier de commande est un fichier texte comportant une ou plusieurs lignes d'instructions. On peut utiliser indifféremment les formats ANSI, Unicode UTF-8BOM (UTF-8 Windows) ou UTF-16LE Unicode. Vous pouvez inclure des commentaires dans un fichier de commande car tout ce qui se situe après un point-virgule sur une ligne est ignoré. Vous pouvez donc mettre en commentaire – et, ainsi, les neutraliser – les lignes dont vous n'avez pas besoin temporairement. Par exemple :

```
MyProg.obj          ; lier ce fichier pour constituer MyProg.exe
;/debug coff        ; utiliser cette ligne pour ajouter une information de débogage
MyProg.res          ; utiliser ce fichier en tant que ressource
```

1.3.1.1 Fichiers d'entrée – fichiers objet et res

Étant donné que la tâche principale du linker est de lier entre eux des fichiers objet, les fichiers d'entrée ont l'extension *obj*. Pour ce faire, il suffit de les énumérer dans n'importe quel ordre. Un fichier *res* (contenant des ressources) peut également être spécifié comme fichier d'entrée.

Notez que, sauf si le commutateur `/fo` est utilisé, l'exécutable prend le nom du premier fichier *obj* ou *res* nommé.

Si le fichier n'est pas dans le répertoire courant, vous devez inclure son chemin pour y parvenir.

1.3.1.2 Fichiers d'entrée – fichiers dll/ocx/exe/drv

Un autre type de fichier d'entrée est celui qui, comme une DLL, contient des *exports*. Les exports peuvent être présents dans les fichiers DLL, OCX, DRV ou autres fichiers EXE et vous pouvez en fournir les noms dans la ligne de commande, dans le fichier de commande ou même dans votre code source en utilisant la directive #dynamiclinkfile si vous utilisez GoAsm (voir le volume 1 de la présente documentation qui en décrit le principe). Contrairement à d'autres linkers, il n'est pas nécessaire d'utiliser de fichiers LIB. Pendant le processus de liaison, GoLink examine la liste des fichiers un par un pour les importations. Lorsqu'il a trouvé toutes les importations requises, il cesse de scruter les fichiers listés. De ce fait, GoLink fonctionnera plus rapidement si vous énumérez ces fichiers par taux d'utilisation décroissant, en mettant en haut de la liste ceux qui contiennent la plupart des importations demandées. Habituellement, l'ordre le plus satisfaisant en matière de classement des DLLs système doit intégrer les considérations suivantes :

Ces DLL contiennent la plupart des API que vous utiliserez :

```
Kernel32.dll
User32.dll
Gdi32.dll
```

Celles-ci contiennent les API des contrôles et dialogues communs que vous pourriez utiliser :

```
COMCTL32.dll
comdlg32.dll
```

Enfin, vous devrez peut-être utiliser également ces DLL :

```
OLEAUT32.dll
Hhctrl.ocx
winspool.drv
shell32.dll
```

Il existe, bien sûr, plusieurs autres DLL système. Vous pouvez identifier la DLL qui héberge l'API que vous souhaitez utiliser en vous référant à la documentation de l'API dans le Kit de développement logiciel (SDK) de Windows, disponible gratuitement auprès de Microsoft. Voir mon site web pour obtenir un lien en ce sens.

Vous pouvez voir combien d'importations de votre programme proviennent de chaque DLL en utilisant le commutateur /files dans la ligne de commande ou un fichier de commande.

1.3.1.3 Fichiers d'entrée 32 et 64 bits

GoLink détecte automatiquement le fait que les fichiers d'entrée soient en 32 ou 64 bits et produit un exécutable en conséquence. Aucune commutation de ligne de commande n'est nécessaire. Dans la mesure où vous ne pouvez pas mélanger les codes 32 et 64 bits, GoLink affichera une erreur si vous vous avisez de contrevenir à ce principe.

Sauf si vous avez désactivé les rapports de la ligne de commande à l'aide des commutateurs /ni ou /no, GoLink vous indiquera dans quel format il a produit l'exécutable.

1.3.1.4 Ordre de recherche

Vous n'avez pas besoin de d'indiquer un chemin d'accès au fichier dll / ocx / exe / drv, car GoLink recherche le fichier selon l'algorithme suivant :

1. dans le répertoire à partir duquel GoLink a été lancé
2. si différent, dans le répertoire courant
3. dans le répertoire système Windows
4. dans le répertoire Windows
5. dans les répertoires mentionnés dans la variable d'environnement PATH

Vous n'aurez normalement pas besoin d'utiliser des versions spécifiques de ces fichiers, mais si vous souhaitez le faire, assurez-vous de donner le chemin d'accès au fichier spécifique ou placez ce dernier dans le répertoire à partir duquel GoLink a été lancé.

Vous pouvez visualiser le chemin d'accès du fichier utilisé, ses date et heure de création et sa taille en utilisant le commutateur /files dans la ligne de commande ou un dans le fichier de commande si tel est le cas. Notez que les dates et les heures sont ajustées pour permettre le réglage de l'heure d'été sur l'ordinateur exécutant GoLink.

1.3.1.5 Sortie d'écran – contrôle, limitation et redirection

Le message de sortie par défaut de GoLink à la console se borne à mentionner le copyright et à mentionner les fichiers de sortie. Il ne s'agit que de quelques lignes, mais s'il y a des erreurs ou si vous utilisez les commutateurs /files ou /unused, la sortie peut être plus prolifique. Vous pouvez définir le contenu du message de sortie en utilisant différents commutateurs, par exemple sur le fait qu'il y ait ou non une sortie et sur la nature des messages de sortie. Vous devrez peut-être le faire si vous utilisez des fichiers make. Voir le descriptif des commutateurs /n pour plus de détails.

Si vous exécutez GoLink à partir d'une fenêtre MS-DOS (invite de commande), alors vous pouvez utiliser le commutateur /more. Si le volume des messages excède en effet la taille de l'écran, ce comutateur le limitera à un plein écran puis vous serez invité à appuyer sur une touche pour afficher la suite. Si vous n'appuyez pas sur une touche (ou que vous donnez une entrée de souris), vous avez un délai de 60 secondes au-delà duquel l'écran suivant s'affichera. Toutefois, si vous désirez examiner des sorties aussi volumineuses à tête reposée, vous avez également la possibilité de les rediriger vers un fichier à l'aide de l'indicateur de redirection DOS :

```
GoLink @command.fil > filename
```

Le format du fichier destination créé par GoLink peut être différent. Les règles suivantes sont appliquées :

- S'il existe un fichier de commande au format Unicode, le fichier destination aura le même format que le premier fichier de commande.
- S'il n'y a pas de fichier de commande dans un format Unicode, le format du fichier destination sera en ANSI si vous travaillez avec W9x ou ME. Avec NT/2000 ou XP, le fichier sera en format UTF-8 Unicode avec BOM[1] (Byte Order Mark). Il s'agit d'autoriser les noms de fichiers Unicode qui pourraient être sur la ligne de commande (et non pas dans un fichier de commande), ainsi que des labels de code et de données dans le fichier objet qui pourraient être en Unicode. Au besoin, faites-moi savoir si cela pose des problèmes.

Vous devez utiliser le commutateur /more plutôt que d'utiliser le filtre "more" de DOS.

1.3.2 Commutateurs de la ligne de commande

Les commutateurs de ligne de commande que vous pouvez utiliser sont les suivants :

/b	**= bip sur erreur.**
/base xxxx	**= définit la base d'image** (en hexa). Par exemple : /base 50000 définit la base d'image attendue à 50000h. Ce commutateur permet de définir l'endroit, dans la mémoire virtuelle, où vous souhaitez que l'exécutable soit chargé (la *base d'image*). Si vous ne définissez pas cette valeur, GoLink utilise une valeur par défaut de 400000h pour les EXE, 10000000h pour les DLL et 10000h pour les pilotes. Vous ne devez normalement pas définir cette valeur lors de la création d'un EXE car chaque EXE chargé a sa propre mémoire virtuelle. Cela signifie qu'à tout moment un certain nombre de programmes peuvent être exécutés, tous avec la même base d'image, et il n'y aura aucun conflit entre eux. Mais si l'EXE s'appuie sur des DLL, le

[1] L'indicateur d'ordre des octets se nomme BOM (Byte Order Mark) en anglais. Cette technique, qui s'apparente à une signature, repose sur l'hypothèse que la séquence d'octets correspondant à l'indicateur d'ordre des octets en UTF-8 a une très faible probabilité d'occurrence dans un texte encodé dans le codage de caractères ancien. C'est notamment le cas en ISO 8859-1. En effet, en UTF-8, l'indicateur d'ordre des octets est codé par la séquence EF BB BF, qui correspond en ISO 8859-1 au texte « ï»¿ ». (source Wikipédia – entrée Indicateur d'ordre des octets)

	chargeur Windows va également charger ces DLL dans la mémoire virtuelle de l'EXE. À l'intérieur de la DLL, GoLink aura écrit certaines adresses en supposant que la DLL serait chargée à cette adresse. Si la DLL ne peut pas être chargée à l'adresse de base attendue (car elle est occupée par une autre DLL), le chargeur devra la charger à une autre. Si cela se produit, le chargeur devra écrire sur les adresses insérées par GoLink. Ces adresses nécessitant un changement sont appelées relocalisations de bases. De toute évidence, ce processus prend du temps pendant le chargement et vous pouvez contribuer à accélérer le processus en spécifiant votre adresse de base pour vos DLL. Cela garantira que la DLL sera effectivement chargée à l'adresse de base attendue. Je suggère d'utiliser la zone 10000000h à 60000000h (le système lui-même a tendance à utiliser la zone en dehors de ces valeurs). Assurez-vous que votre valeur est arrondie pour respecter une modularité de 64 Ko. Vous pouvez utiliser GoBug (commande *inspect/search promenade memory/promenade*) pour savoir où le système d'exploitation charge vos DLL. Il est intéressant de noter que les DLL système elles-mêmes utilisent ce système d'adresse de base décalé. Habituellement, cela se trouve dans la zone de mémoire de 80000000h et plus. GoLink insère l'information de relocation de base dans l'exécutable uniquement si vous utilisez le commutateur `/base xxxx`, le commutateur `/dynamicbase` ou si vous êtes en train de créer une DLL ou un pilote.
/console	**= élabore un exécutable de console.** Ce commutateur positionne un flag dans l'exécutable indiquant au chargeur Windows que l'exécutable ne dépendra pas de l'interface utilisateur graphique de Windows (Windows GUI ou Graphic User Interface). En pratique, cela signifie que l'exécutable ne recevra pas de file d'attente de messages et qu'il ne nécessitera pas de mémoire pour créer des fenêtres et les contextes de périphérique résultants. D'autre part, l'exécutable sera automatiquement muni d'une "console" par le chargeur, permettant l'entrée et la sortie de divers périphériques connectés à l'ordinateur. En raison de leur moindre complexité, les applications de console ont tendance à charger et à sortir plus rapidement que les applications graphiques.
/debug coff	**= ajoute des informations de débogage au format coff intégrées à l'exécutable final.** La sortie de débogage de GoLink est adaptée à ce qui est habituellement requis par les programmeurs en assembleur. Fondamentalement, chaque symbole connu du linker est placé dans l'information de débogage avec sa valeur. Cela peut ensuite être utilisé pour faire vivre le programme sous contrôle de débogage, en utilisant GoBug ou d'autres débogueurs qui acceptent les informations de débogage au format coff.
/debug dbg	**= place les informations de débogage dans un fichier dbg.** Si ce commutateur est utilisé, GoLink place les informations de débogage au format coff dans un fichier .dbg dans un sous-dossier "exe" ou "dll" selon le type d'exécutable en cours. Le sous-dossier est utilisé pour s'assurer qu'il n'y a pas de confusion entre l'EXE principal et ses DLL. Ainsi, par exemple, MyProg.exe aura-t-il ses informations de débogage placées dans exe\MyProg.dbg tandis que MyProg.dll les aura dans dll\MyProg.dbg.
/dll	**= élaboration d'une DLL.** GoLink produit une DLL si ce commutateur est utilisé ou si vous utilisez le commutateur /fo en spécifiant un fichier de sortie avec les extensions DLL ou CPL. Si aucune de ces conditions ne s'applique et vous ne faites pas de pilote, GoLink élabore un EXE. GoLink insère les informations de relocation de base dans l'exécutable uniquement si vous utilisez le commutateur /base xxxx, le commutateur /dynamicbase ou si vous créez une DLL ou un pilote.
/driver	**= élabore un pilote.** GoLink élabore un pilote si ce commutateur est utilisé ou si vous utilisez le commutateur /fo et que vous spécifiez un fichier de sortie avec l'extension SYS. GoLink ne définit pas le flag WDM, à moins que /wdm ne soit spécifié. Consultez également sur ce point les sections création d'un driver selon le Modèle de Driver Windows et spécification du flag uponly.

	GoLink insère les informations de relocation de base dans les drivers.
/dynamicbase	**= spécifie que l'exécutable peut être rebasé au moment du chargement en utilisant la Disposition Aléatoire de l'Espace d'Adressage ou ASLR (Address Space Layout Randomization).** Cela définit le flag IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE. Les versions du système d'exploitation et du sous-système sont définies sur 5.1. GoLink insère également des informations de relocation de base dans l'exécutable.
/e	**= fichier de sortie vide autorisé** À l'aide de ce commutateur, vous pouvez créer un fichier PE "shell" qui contiendra toutes les parties formelles nécessaires, mais pas de section.
/entry xxxx	**= fixe le point d'entrée du programme sur xxxx.** Par exemple, `/entry main` démarrera le programme au label de code "main:". Si vous ne spécifiez pas le point d'entrée avec ce commutateur, GoLink suppose que `START` sera utilisé (quelle que soit la casse des caractères). Donc, si vous souhaitez utiliser cette valeur par défaut, vous devez vous assurer que votre code comporte le label `"START:"` au bon endroit pour commencer l'exécution. Chaque fichier EXE nécessite un point d'entrée, mais dans les DLL, ce dernier est facultatif. GoLink vous avertit s'il ne peut pas trouver le point d'entrée dans votre fichier exécutable. Pour la compatibilité avec les outils Microsoft, GoLink reconnaît également le point d'entrée spécifié au moment de l'assemblage en utilisant END. Cette information est transmise au linker à l'aide de la section `.drectve`.
/export xxxx	**= spécifie les fonctions, les données et les ordinaux exportés.** Voir le paragraphe Export des procédures et données.
/files	**= rapport détaillé sur les fichiers utilisés et créés.** Cette option indique les chemins d'accès complets, les dates, heures et taille des fichiers utilisés et des fichiers de sortie lors du processus d'édition de liens. Il existe d'autres informations utiles telles que le nombre d'importations, d'exportations et de symboles de débogage. Notez que tous les indications de date et heure des fichiers sont ajustées sur celles de l'ordinateur exécutant GoLink.
/fo	**= spécifie le fichier sortie ainsi que son chemin d'accès.** Si le fichier de sortie et son chemin d'accès ne sont pas spécifiés, GoLink suppose que le fichier de sortie doit être un EXE et doit avoir le même nom et le même chemin d'accès que le premier fichier objet ou de ressources (res) fournis à l'entrée. Vous pouvez utiliser le commutateur /fo pour modifier ces valeurs par défaut. Par exemple : `/fo MyProg.exe` ou `/fo MyProg.dll` ou `/fo c:\exe\MyProg.exe`
/h ou **/?**	**= affichage à l'écran des différentes options permises par GoLink.**
/heapinit xxxx	**= spécifie la taille d'engagement du tas (heap).** Si vous ne spécifiez pas ce commutateur, la valeur de 10000h (64 Ko) est utilisée par défaut.
/heapsize xxxx	**= spécifie la taille de la réserve de tas (heap).** Si vous ne spécifiez pas ce commutateur, une valeur de 100 000h (1 Mo) est utilisée par défaut.
/largeaddressaware	**= spécifie que l'exécutable peut gérer des adresses supérieures à 2 Go.** Active le flag IMAGE_FILE_LARGE_ADDRESS_AWARE. Si le commutateur /base n'est pas spécifié, GoLink utilise une Base d'Image par défaut de 14000000h pour les EXE et de 180000000h pour les DLL. Pour un exécutable 64 bits, si ce commutateur

	est utilisé avec /dynamicbase, un autre flag est également défini pour l'ASLR 64 bits HIGHENTROPYVA.
/mix	**= lie les fichiers d'entrée et le code lib produit par un compilateur qui enrichit les symboles.** Une partie des empreintes caractéristiques d'un compilateur "C", et de l'assembleur MASM, est l'enrichissement des symboles dans les fichiers objet qu'ils produisent. Les exemples sont les suivants : _MessageBoxW@16 (*call indirect à une API avec 4 paramètres*) __imp__GetKeyState@4 (*call direct à une API avec 1 paramètre*) _MyDataLabel (*enrichissement simple appliqué à un label de donnée*) _MyCodeLabel (*enrichissement simple appliqué à un label de code*) _MyCodeLabel@4 (*enrichissement appliqué à un label de code déclaré comme ayant un paramètre*). L'idée derrière l'enrichissement de la forme *_Name@xx* est qu'il fournit une vérification que le nombre de paramètres déclarés pour une fonction est identique entre les différents modules (fichiers objet). En pratique cependant, l'accroissement de complexité de code source induit (par exemple, dans MASM, la nécessité pour EXTRN ou PROTO de définir le type de symbole) surpasse largement les avantages résultants. Certains compilateurs utilisent également l'enrichissement pour améliorer les rapports d'erreur, mais cela pose un autre problème : il n'y a pas de format cohérent pour cela. Et comme GoAsm ne pratique pas l'enrichissement des symboles (sauf si le commutateur GoAsm `/ms` est utilisé), GoLink ne s'attend effectivement pas à ce que les symboles le soient. Cela signifie que si vous souhaitez utiliser GoLink pour lier des fichiers produits par un compilateur tiers qui enrichit les symboles, vous devez spécifier le commutateur `/mix` qui fera en sorte que GoLink enlève l'enrichissement lors de la comparaison des noms de symboles. Si vous utilisez le commutateur `/mix`, vous ne pourrez pas utiliser intentionnellement un soulignement d'entête pour différencier des symboles. Par exemple, GoLink considérera que _DOG et DOG correspondent en fait au même nom. Si le commutateur `/mix` est utilisé, les symboles des types IMAGE_SYM_CLASS_STATIC et IMAGE_SYM_CLASS_LABEL seront limités au module (fichier objet) dans lequel ils apparaissent, ainsi que le prévoient les compilateurs tiers. Sinon, si ces types ont le même nom, GoLink détectera des symboles en double. Le commutateur `/mix` peut également résoudre des problèmes si vous avez fusionné, dans une bibliothèque statique de module GoAsm, du code réalisé avec ces outils tiers. Si un tel code appelle ou fait référence à un autre code fusionné, il est possible que cela se fasse avec des symboles enrichis. Ceux-ci ne seraient pas reconnus par GoLink à moins que le commutateur `/mix` ne soit utilisé.
/more	**= affiche l'écran de console puis attend la saisie.** Voir la section Sortie d'écran – contrôle, limitation et redirection
/ne **/ni** **/nw** **/no**	**= interdit tout message d'erreur.** **= interdit tout message d'information.** **= interdit tout message d'avertissement.** **= interdit tout message de sortie.** Si vous utilisez un fichier make, vous pouvez supprimer les messages de sortie et ne compter que sur le code de sortie de GoLink (erreur = 1), ou vous pouvez simplement réduire les informations de sortie. Dans un fichier batch, vous pouvez utiliser le retour d'erreur avec ERRORLEVEL, par exemple, le traitement décrit ci-dessous s'arrêtera s'il y a un retour d'erreur : `GoLink @command.fil` `IF ERRORLEVEL 1 PAUSE` Les messages d'erreur dont l'énumération suit sont ceux qui empêcheront GoLink de créer l'exécutable : - mémoire insuffisante (*Insufficient memory*) - utilisation incorrecte de commutateurs de ligne de commande (*Incorrect use of command line switches*) - impossibilité d'ouvrir ou de lire un fichier spécifié (*unable to open or read a specified file*)

	- fichiers au mauvais format ou corrompus (*Files in wrong format or corrupted*) - information incorrecte fournie pour le fichier de sortie (*incorrect information given for output file*) - impossible de créer un fichier de sortie (*unable to make output file*) - point d'entrée spécifié non trouvé ou absent d'une section de code (*specified entry point not found or not in a code section*) - duplication Resource Type, nameID ou Language trouvés dans le fichier de ressource RS (*duplicate Resource Type, nameID or Language found in res file*) - pas de numéro ordinal défectueux ou en double si import/export direct par ordinal (*no defective or duplicate ordinal number if direct import/export by ordinal*) - symboles déclarés plus d'une fois (*symbols declared more than once*) - symboles non déclarés du tout (*symbols not declared at all*) - symboles exportés non trouvés (*exported symbols not found*) Les messages d'information se composent du copyright et des résultats en sortie de traitement. Les messages d'avertissement n'arrêtent pas l'exécutable en cours. Ils incluent le point d'entrée introuvable et les cas où une DLL n'a fait l'objet d'aucun export. Voir aussi la section Sortie d'écran – contrôle, limitation et redirection.
/nxcompat	**= spécifie que l'exécutable est compatible avec le mécanisme de Prévention de l'Exécution des Données** (Data Execution Prevention ou DEP)**.** Active le flag IMAGE_DLLCHARACTERISTICS_NX_COMPAT. Les versions du système d'exploitation et du sous-système sont définies sur 5.1.
/osversion xxxx	**= définit le numéro de version majeur et mineur du système d'exploitation.** Il s'agit d'une valeur hexadécimale de 32 bits avec une valeur par défaut de 00040000 pour la version 4.0 ajustée à la version 5.1 avec l'utilisation des commutateurs /nxcompat ou /dynamicbase ou la version 5.2 pour un fichier 64 bits.
/stackinit xxxx	**= engagement initial de la pile lors du démarrage ou sur les nouveaux threads.** Ici, vous pouvez spécifier la taille en octets (en utilisant l'hexadécimal) de la quantité de mémoire que le système doit réserver pour la pile lorsque l'application démarre ou lors d'un nouveau thread. Si vous n'utilisez pas ce commutateur, une valeur de 10000h (64 Ko) est allouée par défaut. Indépendamment de la valeur spécifiée à l'aide de ce commutateur, le système augmente toujours la quantité de mémoire engagée dans la pile dans des blocs de 4K comme l'application l'exige. Ce commutateur peut être utilisé si votre application doit créer de l'espace pour une grande quantité de données locales. Sans lui, une exception pourrait se produire si vos données locales sont supérieures à 4K. La raison en est que le système utilise la Gestion des Exceptions Structurée (*Structured Exception Handling* ou SEH) pour gérer l'agrandissement de la pile au fur et à mesure de son besoin, et cela en établissant des pages de garde juste à côté et plus profondément dans la zone de pile déjà engagée. Si votre application tente d'écrire ou de lire à partir d'un endroit au-delà de ces pages de garde, une véritable exception se produira plutôt qu'un signal invitant le système à créer plus de pile. Selon nos expérimentations, la taille des pages de garde semble varier, mais il est peu probable qu'elle soit jamais inférieure à 4 K qui est la taille normale d'une page.
/stacksize xxxx	**= taille d'allocation de pile au démarrage ou sur de nouveaux threads.** Ici, vous pouvez spécifier la taille en octets (en utilisant l'hexadécimal) de la zone de mémoire que le système doit attribuer pour l'utilisation potentielle de la pile, si l'application l'exige. Si vous n'utilisez pas ce commutateur, une valeur de 100000h (1 Mo) est allouée par défaut. Au démarrage, le système alloue simplement l'espace dans la mémoire virtuelle et seulement une petite quantité est engagée au début. Le système engage le reste automatiquement par blocs de 4K si l'application l'exige. La valeur donnée avec ce commutateur s'applique à la fois au thread principal de l'application et aux nouveaux threads qu'elle constitue. Ce commutateur peut être utilisé si votre application nécessite plus de 1 Mo d'espace de pile. La spécification de plus de 1 Mo n'utilise pas la mémoire inutilement puisque celle-ci n'est engagée que si elle est effectivement utilisée. Normalement, 1 Mo est plus que suffisant pour la pile, mais vous pourriez avoir besoin de ce commutateur si vous conservez de grandes quantités de données locales sur la pile dans des procédures hautement récursives.

/subsystemversion xxxx	**= définit le numéro de version majeur et mineur du sous-système.** Il s'agit d'une valeur hexadécimale de 32 bits avec une valeur par défaut de 00040000 pour la version 4.0, ajustée à la version 5.1 avec l'utilisation de /nxcompat ou /dynamicbase, ou la version 5.2 pour un fichier 64 bits.
/unused	**= rapport sur les labels non utilisés / non référencés.** *Ce commutateur ne fonctionne qu'avec les fichiers objet GoAsm ou les fichiers objet compatibles "Go"*. Voir la note technique à ce sujet. S'il est utilisé, GoLink rendra compte des labels de donnée et de code qui n'ont pas été référencées de quelque manière que ce soit. De telles déclarations de donnée ou de code redondantes sont facilement oubliées dans le code source par erreur lors du développement d'un programme. Maintenant, il vous est possible de les identifier et les éliminer.
/uponly	**= spécifie le flag uponly.** Active le flag IMAGE_FILE_UP_SYSTEM_ONLY qui indique à Windows que le fichier (habituellement un pilote) ne peut être utilisé que sur des machines à processeur unique.
/version xxxx	**= définit la version de l'image en une valeur majeure/mineure.** Il s'agit d'une valeur hexadécimale de 32 bits. Par exemple, la version 20001 définit la valeur de la version principale à 2 et la valeur de la version mineure à 1.
/wdm	**= élabore un pilote de type Windows Driver Model.** GoLink effectue un pilote WDM si ce commutateur est utilisé. Il s'agit d'un pilote conçu pour plusieurs plates-formes Windows. Il n'est pas nécessaire de spécifier le commutateur `/driver` également. Voir aussi les sections Création d'un pilote et Spécification d'un flag.

1.4 Import et export de fonctions et de données

1.4.1 Import de données

Les données ne peuvent être importées qu'*indirectement*. Cela signifie que vous ne pouvez importer qu'un *pointeur* vers les données. Cependant, en utilisant ce pointeur, vous pouvez obtenir les données elles-mêmes.

Par exemple, en supposant que DATA_VALUE soit un export de données dans un autre programme dans GoAsm, vous obtenez le pointeur et les données comme suit :

```
MOV EBX,[DATA_VALUE]        ; récupération du pointeur de DATA_VALUE
MOV EAX,[EBX]               ; récupération de la valeur proprement dite
```

De la même manière que pour importer des procédures à partir d'autres programmes au moment de l'édition des liens, vous donnez à GoLink le nom de l'exécutable contenant l'import.

1.4.1.1 Import direct par ordinal

La méthode habituelle d'importation d'une fonction dans une DLL s'effectue à l'aide d'un CALL *procedure*, qui importera la procédure par son *nom*. Ce qui se passe ici, c'est que lorsque le chargeur Windows démarre le programme, il recherche les DLL pour les importations requises par le programme. Cela se fait en comparant les noms des exportations DLL par rapport aux noms des importations du programme. Pour accélérer ce processus avec des DLL *privées*, il arrive que l'exportation et l'importation par ordinal soient utilisées. Alors, le chargeur peut trouver l'importation correcte en utilisant un index dans une table dans la DLL. Notez qu'il est imprudent d'utiliser cette méthode pour les DLL du système Windows, car la pérennité des valeurs ordinales des exportations n'est absolument pas garantie au fur et à mesure que les DLL font l'objet de mises à jour.

En utilisant GoLink et son programme complémentaire GoAsm, vous pouvez importer par ordinal en utilisant cette syntaxe simple dans le script source de l'assembleur :

```
CALL MyDll:6
```

Cette instruction appelle la procédure numéro 6 dans MyDll.dll. La façon dont cela fonctionne est simple : GoAsm passe le symbole "MyDll:6" à GoLink dans le fichier objet. GoLink sait que c'est une importation par ordinal en raison de la présence des deux points. Notez que l'extension "dll" est implicite si aucune extension n'est donnée. Supposons que vous souhaitiez qu'une DLL appelle une fonction dans l'exécutable principal par ordinal. Alors vous pouvez utiliser :

```
CALL Main.exe:15
```

L'instruction ci-dessus appelle la 15ème fonction dans Main.exe.

Les appels vers les ordinals utilisant la forme absolue (en utilisant les opcodes FF15) résulteront de l'utilisation de cette syntaxe :

```
CALL [Main.exe:15]
```

Vous ne devez pas inclure le chemin d'accès du fichier dans l'appel. GoLink recherchera le fichier en utilisant le même ordre de recherche que lors de la recherche de DLL. S'il s'avérait toutefois nécessaire de fournir un chemin d'accès, il devrait être fourni dans la ligne de commande ou dans un fichier de commande plutôt qu'incorporé dans l'appel dans le script d'assemblage.

Si vous n'utilisez pas GoAsm, vous pourrez toujours utiliser cette méthode d'importation par ordinal, pourvu que votre assembleur autorise un symbole contenant deux points (et un point si vous devez ajouter l'extension).

De toute évidence, pour utiliser cette méthode d'appel d'une fonction par ordinal, vous devez vous assurer que le nombre ordinal de la fonction est fixe, voir la section Export par ordinal.

1.4.1.2 Import par une DLL spécifique

Si des fonctions portant le même nom sont présentes sur deux DLL ou plus (parmi celles spécifiées au linker au moment de l'édition de liens proprement dite), il vous est possible de forcer le linker à utiliser une importation à partir d'une DLL nommément désignée.

Vous devrez peut-être le faire pour passer outre un appel d'API lors de l'utilisation de Microsoft Layer for Unicode en utilisant GoLink avec le commutateur `/mslu`. Cela ajoutera un code de chargeur spécial à votre fichier exécutable qui fera en sorte que votre application appelle l'API concernée dans unicows.dll au moment de l'exécution si elle s'exécute sous Windows 95, 98 ou ME. Si vous avez écrit votre propre wrapper autour d'une API particulière, vous devrez peut-être arrêter cela. Vous pouvez donc forcer GoLink à lier à une DLL particulière en utilisant la syntaxe `NameOfDll:NameOfAPI`.

Par exemple, si vous souhaitez ignorer *EnableWindow* dans unicows.dll en toutes circonstances, et toujours lui préférer *EnableWindow* dans User32.dll, vous utiliserez :

```
User32:EnableWindow
```

1.4.2 Export de procédures et de données

Vous pouvez mettre vos procédures et vos données à la disposition d'autres exécutables en les exportant. Dans Windows, il est habituel que les DLL soient utilisées pour les exportations, mais parfois une DLL devra appeler une procédure ou utiliser des données dans un fichier EXE, et dans ce cas, le fichier EXE exportera. L'exportation peut être effectuée soit au moment de l'édition de liens (vous indiquez à GoLink les symboles à exporter), soit au moment de l'assemblage en utilisant GoAsm. Dans ce dernier cas, GoAsm indique alors à GoLink les informations d'exportation via la section .drectve dans le fichier objet. Consultez le volume 1 de cette documentation, consacré à GoAsm, sur la manière de déclarer les exportations au moment de l'assemblage.

Il est facile de déclarer les exportations au moment du lien. Vous utilisez le commutateur `/export` ou `/exports` suivi d'un ou plusieurs noms d'exportation. À moins d'utiliser le caractère de continuation \, vous devrez utiliser le commutateur à nouveau si vous devez démarrer sur une autre ligne.

Exemple :

```
/EXPORT LOAD_ELEMENT
/EXPORTS CALCULATE, ADJUST_DATA, DATA_VALUE
```

Ici, trois fonctions sont exportées, puis un pointeur de données est exporté à son tour. Seul le nom du symbole est nécessaire – il n'est pas nécessaire de préciser à GoLink l'appartenance du symbole à une section de code ou de données.

1.4.2.1 Export par ordinal

Normalement, les exports sont effectués *par leur nom*. Ce qui se passe ici, c'est que lorsque le chargeur Windows démarre le programme, il recherche les DLL pour les importations requises par le programme. Cela se fait en comparant les noms des exportations DLL par rapport aux noms des importations du programme. Pour accélérer ce processus avec des DLL *privées*, parfois l'exportation et l'importation par ordinal sont utilisées. Ensuite, le chargeur peut trouver l'importation correcte en utilisant un index dans une table dans la DLL. Notez qu'il est imprudent d'utiliser cette méthode pour les DLL système de Windows, car les valeurs ordinales des exportations sont susceptibles d'être modifiées à l'occasion des changements de version des DLL.

Afin d'utiliser cette méthode avec toutes les garanties de sécurité, il est impératif que le programme d'exportation spécifie une valeur ordinale pour une exportation particulière et que le linker ne puisse changer cela.

Cela peut se faire très facilement. Par exemple :

```
/EXPORTS CALCULATE:2, DATA_VALUE:6
```

Ici, GoLink est invité à utiliser les ordinaux 2 et 6 pour les exportations.

1.4.2.2 Export anonyme par ordinal

L'export par ordinal n'empêche pas le nom de l'export de figurer dans l'exécutable final. C'est parce que c'est le programme d'importation qui décide s'il faut importer par ordinal ou par nom. Tout ce que le programme d'exportation peut faire est donc de fixer la valeur ordinale. Cependant, quelquefois un programmeur pourrait souhaiter qu'aucun nom pour l'exportation ne figure dans l'exécutable final. Vous voyez parfois ces exportations «sans nom» dans les DLL du système par exemple, probablement pour cacher le travail effectué par des fonctions particulières. Pour ce faire, ajoutez le mot NONAME à la fin de l'exportation. Par exemple :

```
/EXPORTS CALCULATE:2:NONAME, DATA_VALUE:6:NONAME
```

Ici, la valeur du label de code CALCULATE sera exportée en tant que numéro ordinal 2, mais le nom de l'exportation n'apparaîtra pas dans l'exécutable final. Cela signifie aussi que si un autre programme essayait d'appeler la fonction CALCULATE, il échouerait. La fonction ne peut être appelée que par ordinal. Cela fonctionne également pour les données, comme dans cet exemple.

1.4.3 Utilisation d'Unicode pour l'import ou l'export de labels

GoLink est susceptible de pouvoir accueillir des labels Unicode dans les fichiers objet, et il s'attend à ce que ce soit soit au format UTF-8 (voir le chapitre). Lorsque vous créez l'exécutable et créez des informations sur les symboles de débogage, GoLink conserve ce format. Cela signifie que vous pouvez importer et exporter des fonctions et des données à l'aide d'Unicode (vous pouvez utiliser des caractères non-romains dans vos symboles si vous le souhaitez). Notez que les IDs de ressources et les ressources nommés sont traités différemment. Ils sont toujours au format UTF-16, comme l'exigent les spécifications du format PE.

1.5 Gestion d'Exception, SafeSEH et IMAGE_LOAD_CONFIG_DIRECTORY

1.5.1 Gestion des Exceptions 32 bits et SafeSEH

GoLink traite les informations d'index de la table de symboles SafeSEH fournies dans les sections .sxdata[2] en créant un tableau d'adresse virtuelle relative (RVA) trié des procédures de traitement d'exception données, fournit une structure IMAGE_LOAD_CONFIG_DIRECTORY32 par défaut et effectue les ajustements appropriés pour la structure IMAGE_OPTIONAL_HEADER.

Vous pouvez également fournir manuellement cette structure et la table associée comme suit à l'aide d'un symbole appelé *__load_config_used* (notez les 2 symboles de soulignement dans l'entête) que GoLink reconnaît pour effectuer les ajustements appropriés à IMAGE_OPTIONAL_HEADER dans les fichiers 32 bits :

```
CONST     SECTION 'const$~' ALIGN 4
__load_config_used:                    ; IMAGE_LOAD_CONFIG_DIRECTORY32
                        DD    48h      ; Size
                        DD    0        ; TimeDateStamp
                        DW    0        ; MajorVersion
                        DW    0        ; MinorVersion
                        DD    0        ; GlobalFlagsClear
                        DD    0        ; GlobalFlagsSet
                        DD    0        ; CriticalSectionDefaultTimeout
                        DD    0        ; DeCommitFreeBlockThreshold
                        DD    0        ; DeCommitTotalFreeThreshold
                        DD    0        ; LockPrefixTable
                        DD    0        ; MaximumAllocationSize
                        DD    0        ; VirtualMemoryThreshold
                        DD    0        ; ProcessHeapFlags
                        DD    0        ; ProcessAffinityMask
                        DW    0        ; CSDVersion
                        DW    0        ; Reserved1
                        DD    0        ; EditList
                        DD    0        ; SecurityCookie
```

[2] Les sections .sxdata sont une composante du format PE. Selon le site Microsoft, les gestionnaires d'exception valides d'un objet sont répertoriés dans la section .sxdata de cet objet. La section est marquée IMAGE_SCN_LNK_INFO. Elle contient l'indice de symbole COFF de chaque gestionnaire valide, en utilisant 4 octets par index.

```
                        DD    __safe_se_handler_table          ; SEHandlerTable
__safe_se_handler_count   DD    1       ; SEHandlerCount

__safe_se_handler_table:
                      ; la table des gestionnaires RVA vient ici ou est ajoutée ici
                      ; comme indiqué ci-dessous

CODE      SECTION
          ALIGN     8
eHandler:
                              ;
                              ;
          ret

CONST     SECTION 'const$~' ALIGN 4              ; l'alignement est important ici pour les
                                                 ; projets comportant plusieurs fichiers source
          DD        eHandler - Start + 1000h     ; RVA
```

Notez que dans le calcul RVA ci-dessus, Start doit être le premier label au début de la section CODE qui a un RVA de départ par défaut de 1000h. Il est important de le faire de cette façon au lieu de la base d'image (eHandler–400000h) pour s'assurer qu'il n'y aura pas d'entrée de relocation qui jetterait cette figure dans le cas d'une DLL ou que vous utilisiez le commutateur `/dynamicbase` sur la ligne de commande.

Il est également important d'utiliser un nom de section similaire à 'const $ ~' où le texte après le $ est utilisé par le linker pour trier cette section dans la section 'const', avec l'utilisation de '~' en le plaçant à la fin où l'on s'attend à ce que le même ou d'autres fichiers OBJ utilisent la méthode habituelle `.sxdata`. Le placement à la fin de la section permet à GoLink de combiner et trier la table RVA fournie manuellement avec celle fournie à partir du traitement `.sxdata` habituel.

1.5.2 Gestion des Exceptions 64 bits

Une méthode d'organisation par tables très différente est utilisée ici avec des informations de déroulement et de gestion des exceptions fournies dans les sections `.pdata` et `.xdata`. GoLink effectue les ajustements appropriés à IMAGE_OPTIONAL_HEADER et place les informations .xdata dans une section en lecture seule. Les exécutables 64 bits n'utilisent pas SafeSEH. Cependant, vous pouvez toujours fournir manuellement une structure IMAGE_LOAD_CONFIG_DIRECTORY64 comme suit en utilisant un symbole nommé *_load_config_used* (notez qu'il n'y a plus qu'un seul symbole de soulignement en guise d'entête) que GoLink reconnaît pour effectuer les réglages appropriés sur IMAGE_OPTIONAL_HEADER dans les fichiers 64 bits:

```
CONST     SECTION
          ALIGN     8
_load_config_used:  ;IMAGE_LOAD_CONFIG_DIRECTORY64
          DD        70h         ; Size
          DD        0           ; TimeDateStamp
          DW        0           ; MajorVersion
          DW        0           ; MinorVersion
          DD        0           ; GlobalFlagsClear
          DD        0           ; GlobalFlagsSet
          DD        0           ; CriticalSectionDefaultTimeout
          DQ        0           ; DeCommitFreeBlockThreshold
          DQ        0           ; DeCommitTotalFreeThreshold
          DQ        0           ; LockPrefixTable
          DQ        0           ; MaximumAllocationSize
          DQ        0           ; VirtualMemoryThreshold
          DQ        0           ; ProcessAffinityMask
          DD        0           ; ProcessHeapFlags
          DW        0           ; CSDVersion
          DW        0           ; Reserved1
          DQ        0           ; EditList
          DQ        0           ; SecurityCookie
          DQ        0           ; SEHandlerTable - non utilisé
          DQ        0           ; SEHandlerCount - non utilisé
```

1.6 Spécifications concernant le commutateur /unused

Le commutateur /unused dans GoLink identifie et restitue les labels de données et de code qui ne sont pas référencés. Entrent principalement dans cette catégorie les labels inutilisés qui subsistent au terme du processus de développement. Il n'est pas rare de les trouver associés à du code et des données opérationnels bien que devenus sans intérêt et occupant inutilement l'espace du programme. Avec ce commutateur, il devient possible de les identifier et de les éliminer.

Pour permettre à l'éditeur de liens de fournir cette information, il est nécessaire que l'assembleur (ou le compilateur) et l'éditeur de liens travaillent ensemble. C'est parce que l'assembleur (ou le compilateur) ont souvent déjà des références réparties sur les labels dans le fichier objet. Par exemple, dans ce programme simple :

```
CALCULATE:
ADD EAX,EDX
RET
;
START:
CALL CALCULATE
RET
```

Le CALL à la fonction CALCULATE sera codé comme un appel relatif, et puisque le CALL lui-même et sa destination sont dans la même section, l'assembleur connaîtra et pourra appliquer la distance correcte pour l'exécution à transférer. Cela signifie qu'il n'y aura pas de relocation COFF en relation avec le CALL dans le fichier objet et que le linker n'aura aucun correctif à effectuer. À l'inverse, cela signifie que le linker n'aura aucun moyen de savoir si l'étiquette CALCULATE a été référencée ou non.

Dans les fichiers objet de format "Go", tout symbole référencé et traité par l'assembleur ou le compilateur a le flag 2000h défini à + 0Eh dans la table des symboles (c'est le champ "type" dans cette table). Si GoLink trouve ce flag défini, le symbole concerné ne sera pas signalé comme non référencé. En conséquence, l'information de l'assembleur (ou du compilateur) garantit que l'information de GoLink selon laquelle un label n'est pas utilisé s'avère exacte.

Ce flag peut également être utilisé par l'assembleur ou le compilateur pour s'assurer que certains labels sont omis dans la fonctionnalité /unused de GoLink. Par exemple, GoAsm considère chaque membre nommé d'une structure comme une étiquette à part entière. Comme il est probable qu'un certain nombre de ces membres ne sont pas directement référencés, ils apparaîtront dans le rapport inutilisé à moins qu'ils ne soient supprimés. En conséquence, GoAsm affiche tous les membres de la structure en tant que flag 2000h (mais pas le label de la structure elle-même) pour supprimer ces étiquettes dans le rapport /unused.

En pratique, il est également nécessaire pour l'assembleur ou le compilateur d'indiquer à GoLink que le fichier objet est en format "Go". Ceci est réalisé en s'assurant que le premier symbole (c'est-à-dire le premier enregistrement de la table de symboles dans le fichier objet) a le nom du symbole ".GoAsm" (sans les guillemets mais avec le point) avec un nombre de section de –2 et une classe de stockage de 67h. GoLink sait alors qu'il peut donner un rapport précis des labels inutilisés si le commutateur /unused est défini par l'utilisateur.

GoLink ne cherche pas de symboles dans les fichiers *.res* ou *.obj* qui contiennent uniquement des informations sur les ressources. Il n'est donc pas nécessaire de modifier les formats de ceux-ci.

1.7 Aspects juridiques

1.7.1 Copyright

1.7.2 Licence et distribution

Vous pouvez utiliser GoLink gratuitement pour n'importe quel usage. Vous pouvez redistribuer GoLink librement mais sans exiger la moindre rétribution en contrepartie. Par extension, GoLink ne peut faire partie d'un ensemble proposé à la vente. Vous n'avez pas le droit de cacher ou de refuser mes droits d'auteur.

1.7.3 Avertissement

L'auteur a tout mis en œuvre pour que les affichages et fichiers produits par GoLink soient précis, mais vous l'utilisez entièrement à vos risques et périls. L'auteur n'accepte aucune responsabilité découlant d'un fonctionnement incorrect, d'un résultat erroné ou d'erreurs entachant le présent manuel.

1.8 Remerciements

Mes plus chaleureux remerciements à Wayne J. Radburn, de Gatineau, au Québec, en particulier pour ses récentes corrections du programme source, et à Anthony Williams, de Brixham, Devon England, auteur d'ALINK. Et merci pour vos idées, votre soutien et vos rapports de bug et, notamment, Leland M. George de West Virginia, Daniel Fazekas de Budapest, Brian Becker et certains membres du forum *GoAsm Assembler and Tools*.

Chapitre 2 – Éditeur de ressources GoRC

2.1 Introduction

Ce chapitre s'intéresse au compilateur de ressources GoRC et part du principe que le lecteur dispose d'une connaissance minimum des ressources, de leur raison d'être et de leur fonctionnement.

Il existe de légères différences de syntaxe entre le compilateur de ressources Microsoft (*rc.exe*) et le compilateur GoRC. D'une manière générale, GoRC supporte tous les scripts de ressources *.rc* acceptés par le compilateur de ressources Microsoft tout en proposant une syntaxe légèrement plus souple.

Les fichiers source RC (ceux dotés d'une extension ".rc") doivent être composés de texte sans code de formatage, autre que le retour chariot, le saut de ligne ou les tabulations à l'image de ceux qui sont produits par la plupart des éditeurs de texte simples. Des traitements de texte plus sophistiqués peuvent également produire de tels fichiers s'ils sont paramétrés pour produire des fichiers TXT, ASCII ou ANSI.

Le script de ressources et les fichiers d'inclusion peuvent être au format Unicode. Si c'est le cas, GoRC attend un Indicateur d'Ordre des Octets (Byte Order Mark ou BOM[3]). GoRC peut lire le format UTF-8 Unicode avec BOM, ou le format UTF-16LE Unicode avec BOM.

GoRC peut créer deux types de fichiers de ressources binaires à partir d'un script de ressources. Il peut créer un fichier OBJ au format COFF, forme appropriée pour une édition de liens avec d'autres fichiers OBJ en vue de produire l'EXE ou la DLL finale. Il a également la capacité de produire un fichier RES, qui est un fichier intermédiaire au format binaire. GoRC peut également convertir un fichier RES en un fichier OBJ.
Le fichier OBJ fabriqué par GoRC peut être au format Win32 (par défaut) ou Win64 si vous spécifiez le commutateur `/machine AMD64` ou `/machine X64` dans la ligne de commande.

Voici les règles de syntaxe utilisées dans ce manuel :

- Le texte normal affiche les mots tels qu'ils apparaissent dans votre code source.
- *Le texte en italique ne décrit que le type d'entrée dans votre code source – cherchez la description pour voir ce qui peut être mis là.*
- [Tout texte entre crochets signifie que son contenu est facultatif].

2.2 Initiation au Compilateur de Ressources

Syntaxe de la ligne de commande de GoRC :

GoRC [*commutateurs ligne de commande*] *filename*[*.ext*]

Où,

commutateurs ligne de commande peuvent être l'un des suivants :

/h ou /?	aide
/d	définit un mot
/o	crée un fichier OBJ
/r	crée un fichier RES
/fo	spécifie un fichier de sortie
/machine AMD64	ou
/machine X64	le fichier objet de sortie sera en format 64 bits
/ne	annulation de tout message d'erreur
/ni	annulation de tout message d'information
/nw	annulation de tout message d'avertissement
/nu	pas de message d'avertissement pour un type de ressource défini par l'utilisateur
/no	pas de message de sortie du tout

[3] L'indicateur d'ordre des octets se nomme BOM (Byte Order Mark) en anglais. Cette technique, qui s'apparente à une signature, repose sur l'hypothèse que la séquence d'octets correspondant à l'indicateur d'ordre des octets en UTF-8 a une très faible probabilité d'occurrence dans un texte encodé dans le codage de caractères ancien. C'est notamment le cas en ISO 8859-1. En effet, en UTF-8, l'indicateur d'ordre des octets est codé par la séquence EF BB BF, qui correspond en ISO 8859-1 au texte « ï»¿ ». (source Wikipédia – entrée Indicateur d'ordre des octets)

filename	nom du fichier contenant le script de ressource
ext	extension du fichier contenant le script de ressource

2.2.1 Fichiers d'entrée et de sortie

L'action par défaut de GoRC est **RC > RES** et **OBJ**.

Si aucune extension n'est donnée dans la ligne de commande et que GoRC ne trouve pas de nom de fichier portant l'extension .RC, il en recherche un portant l'extension .RES. Si un fichier .RES est trouvé, il en résulte l'action **RES > OBJ**.

L'action par défaut peut être remplacée pour obtenir **RC > RES** de la manière suivante :

- Spécification d'un fichier de sortie RES à l'aide du commutateur /r
- Utilisation du commutateur /fo et attribution d'un nom de fichier avec l'extension RES

L'action par défaut peut être remplacée pour obtenir RC > OBJ de la manière suivante :

- Spécification d'un fichier de sortie OBJ à l'aide du commutateur /o
- Utilisation du commutateur /fo et attribution d'un nom de fichier avec l'extension OBJ

Vous pouvez garantir que l'action par défaut est **RES > OBJ** en donnant au fichier d'entrée l'extension RES.

Par défaut, le nom de fichier de sortie sera le même que celui du fichier d'entrée, mais vous pouvez utiliser le commutateur /fo pour le modifier. Voici comment procéder :

- utiliser /fo *filename* pour produire les fichiers de sortie *filename*.OBJ et *filename*.RES
- utiliser /fo *filename*.OBJ pour produire le fichier de sortie *filename*.OBJ ; le fichier de sortie RES aura le même nom que le fichier d'entrée
- utiliser /fo *filename*.RES pour produire le fichier de sortie *filename*.RES ; le fichier de sortie OBJ aura le même nom que le fichier d'entrée
- utiliser /fo *filename1*.RES, *filename2*.OBJ pour produire des fichiers de sortie correspondant aux noms de fichiers spécifiés ; ici, les 2 noms de fichier sont spérarés par une virgule.

Par défaut, les fichiers de sortie seront placés dans le même répertoire que le fichier d'entrée. Vous pouvez spécifier un chemin d'accès complet pour les fichiers de sortie lorsque vous utilisez le commutateur `/fo`. Vous pouvez inclure les noms de fichiers entre guillemets pour permettre l'utilisation de noms de fichiers contenant des espaces et autres caractères inhabituels.

GoRC peut fonctionner avec les noms de fichiers d'entrée et de sortie Unicode, mais les extensions utilisées doivent être comme indiqué ci-dessus.

2.2.2 Erreurs et Avertissements

Les erreurs entraînent l'arrêt du compilateur de ressources et se traduisent par l'absence de fichier de sortie. La console affiche ensuite la nature de l'erreur et l'endroit où elle s'est produite. (sauf si le commutateur de ligne de commande `/ne` a été spécifié). Le compilateur de ressources retourne également avec un code de 1, susceptible d'arrêter le processus de construction si vous utilisez un fichier make.

Les avertissements n'arrêtent pas le compilateur de ressources. Un avertissement sera donné (sauf si le commutateur de ligne de commande `/nw` a été spécifié) si :

- Une expression n'a pas pu être évaluée en attendant un *identifiant*. Dans ce cas, le compilateur de ressources traite l'expression comme étant un ID avec un nom. Cependant, cela peut ne pas être votre intention, si, par exemple, vous pensiez avoir fourni une directive `#define` pour l'expression.
- Une expression n'a pas pu être évaluée lorsque vous attendez un *type* de ressource. Dans ce cas, le compilateur de ressource traite l'expression comme définissant une ressource, c'est-à-dire une *ressource définie par l'utilisateur*. Encore une fois, cela peut ne pas être votre intention, donc un avertissement est délivré (sauf si le commutateur de ligne de commande `/nu` a été spécifié).
- Aucune ressource n'est trouvée dans le fichier de ressources. Dans ce cas, aucun fichier de sortie n'est créé.

Un avertissement est également délivré si une expression a été définie plus d'une fois dans la ligne de commande, dans un script source ou dans un fichier d'inclusion non ".h". C'est parce qu'il serait inhabituel de définir une expression plus d'une fois et il se peut, par ailleurs, qu'il s'agisse d'une erreur de programmation. Il est parfaitement possible d'annuler une définition précédente en utilisant `#undef` pour que l'expression puisse être redéfinie. Dans ce cas, aucun avertissement n'est donné. En outre, comme le compilateur de ressources n'a pas toujours besoin de se pencher sur les fichiers ".h" pour évaluer une expression, et parce qu'il est supposé que de tels fichiers ".h" devraient être exempts d'erreurs de cette sorte, aucune vérification des définitions précédentes n'est effectuée dans les fichiers d'entête.

2.3 Directives

Les directives sont des mots dans le script de ressources qui donnent des instructions au compilateur de ressources.

Il existe 3 types de directives :

- directives de définition
- directives d'inclusion
- directives conditionnelles

2.3.1 Les Directives de Définition

2.3.1.1 Evaluation des types

Ce sont des macros `#define` qui évaluent un nombre ou une chaîne entre guillemets :

#define *name value*

A partir de ce point, lorsque *name* est rencontré, dans la suite du script de ressource, il est remplacé par *value*.

Exemples

#define IDC_BUTTON 0x20
#define DLGCONTROL 0x20 | 0x40
#define DLGTEXT "Hello world"
#define OTHER DLGCONTROL + IDC_BUTTON
#define WINNT
(ceci donne la valeur 1 à WINNT qui correspond à TRUE)

Le compilateur GoRC **définit automatiquement RC_INVOKED**, de sorte que vous pouvez utiliser les directives conditionnelles pour ignorer le texte de script de ressource indésirable.

Exemple

#ifndef RC_INVOKED
skipped text
#endif

Vous pouvez définir un *nom* à partir de la ligne de commande :

Exemples

GoRC /d WINVER=400h mainres
Cette instruction compile le script de ressource mainres*, sur la base que WINVER est égal à la valeur hexadécimale 400 tout au long.*

GoRC /d WINNT mainres
Cette instruction compile le script de ressources mainres*, sur la base que WINNT a la valeur 1* (*c'est-à-dire TRUE*).

GoRC /d WINNT=55h /d good=0x44 mainres
Exemple de plus d'une définition.

GoRC /d WINNT=330, good=0x44 mainres
Exemple de plus d'une définition utilisant une virgule.

Pour supprimer la valeur d'une définition et avoir un *nom* indéfini, utilisez `#undef` :

#undef *nom*

À partir de ce point dans le script de ressources, toute valeur antérieure pour le nom donné par `#define` n'est plus utilisée.

Exemples

#undef IDC_BUTTON
#undef DLGCONTROL

2.3.1.2 Non évaluation des types

Il s'agit des macros `#define` qui donnent plusieurs résultats :

#define *name value1 value2 value3*

À partir de ce moment, lorsque *name* est rencontré dans le script de ressource, il est remplacé par *value1 value2 value3*.

Exemple

```
#define GRASSHOPPER Deep Influences On Us
#define FILENAME README
#define EXTENSION TXT
FILENAME.EXTENSION (devient README.TXT)
```

2.3.1.3 Macros avec arguments

Ce sont des macros `#define` qui utilisent les arguments fournis lorsque la macro est utilisée.

Syntaxe de la macro dans laquelle les arguments sont déclarés :

```
#define  name(arg1,arg2,arg3,...)   utilisation d'arguments
```

Syntaxe pour fournir les arguments :

```
name(arg1,arg2,arg3,...)
```

Exemples

```
#define NIM($a,$b) $a+$b
NIM(23,45)
```

(*le résultat est égal à 68*)

```
#define NIMROD($a,$b) #$a
NIMROD (HANG,OVER)
```

(*le résultat est "HANG"*)

Notez que le nombre d'arguments *fournis* doit correspondre exactement au nombre d'arguments *déclarés*, mais il n'est pas nécessaire de tous les *utiliser*. Lorsqu'un argument est déclaré, il ne doit y avoir aucun espace entre le nom de la macro et les arguments, mais il peut y avoir des espaces lorsque l'argument est *fourni* ou *utilisé*.

2.3.2 La directive Include

La directive `#include` enjoint au compilateur de ressources de charger et d'examiner le fichier spécifié :

```
#include path\filename
```

Où,

path\filename peut être, soit :

une chaîne entre guillemets
une chaîne non-encadrée de guillemets
une chaîne encadrée par les caractères '<' et '>'

Le compilateur de ressources recherche le fichier en utilisant le chemin d'accès spécifié. Si aucun chemin n'est spécifié, il le recherche dans le répertoire courant. En cas de nouvel échec, il le recherche alors dans le chemin indiqué par la chaîne d'environnement INCLUDE. Vous pouvez définir cette chaîne à l'aide de la commande SET dans la fenêtre MSDOS ou en appelant l'API *SetEnvironmentVariable*. Vous pouvez également utiliser le panneau de configuration (Système/ Paramètres système avancés/ Variables d'environnement[4]) si votre système d'exploitation le permet, puis procéder à un redémarrage. *Remarque : il peut exister une chaîne d'environnement différente pour chaque dossier ou sous-dossier. Assurez-vous que la chaîne d'environnement que vous souhaitez utiliser se trouve dans le répertoire en cours*.

Pour le compilateur de ressources, il existe deux types de fichiers `#include` :

- Il existe des fichiers "header" avec l'extension de fichier ".h". Ces fichiers ne sont pas traités comme faisant partie du script de ressource, mais sont considérés uniquement comme contenant des directives de définition. Cela permet au compilateur de ressources de travailler plus rapidement car il ne tentera pas de chercher un fichier ".h" à moins d'avoir une expression qu'il essaie d'évaluer. Toutes les instructions de script de ressources ordinaires dans un fichier ".h" seront ignorées par le compilateur de ressources.
  Notez que le compilateur de ressources est lui-même conscient de la plupart, voire de la totalité, des styles et expressions habituels utilisés dans un script de ressource. Pour cette raison, il ne sera normalement pas

[4] Constaté sous Windows 8.1

nécessaire d'inclure des fichiers d'en-tête. En contrepartie, vous devrez définir vos propres expressions et identifiants si vous souhaitez en utiliser.

- Le deuxième type de fichiers `#include` ne sont pas des fichiers ".h". Ceux-ci sont traités comme faisant partie du script de ressources lui-même par le compilateur de ressources. Quand une directive `#include` est trouvée pour un fichier non-".h", le compilateur de ressources porte alors son attention sur le fichier #include et, quand il a fini de le scruter, il retourne au script d'origine là où il s'était arrêté auparavant.

Vous pouvez imbriquer des fichiers `#include`, mais c'est une pratique qu'il convient d'éviter cela autant que possible par souci de clarté.

Priorité de définition

Une directive `#define` dans un fichier d'inclusion ".h" est considérée comme moins prioritaire qu'une directive #define dans la ligne de commande, le script de ressource ou dans un fichier d'inclusion non-".h". Par conséquent, s'il y a un conflit entre directives `#define`, celle du fichier d'inclusion ".h" sera ignorée.

2.3.3 Les directives conditionelles

Avec ces directives, vous pouvez sélectionner, lors de la compilation, la partie du script de ressources ou des directives de définition que vous souhaitez compiler. Cela peut être utile si, par exemple, vous voulez créer différentes versions de votre programme à partir du même script de ressource.

La syntaxe de la structure de base d'une directive conditionnelle dans sa forme la plus simple est la suivante :

```
#if condition
    text A
#endif
```

Ici, si la *condition* est VRAIE, le texte A sera compilé. Si, par contre, la condition est FAUSSE, le compilateur sautera par-dessus le texte A et continuera à compiler à partir du `#endif`.

Vous pouvez ajouter quelque chose à faire si la condition est FAUSSE de la manière suivante :

```
#if condition
    text A
#else
    text B
#endif
```

Ici, si la *condition* est VRAIE, le texte A sera compilé, mais le texte B ne le sera pas. Si, par contre, la *condition* est FAUSSE, le texte A sera sauté, mais le texte B sera compilé.

Le `#endif` indique la fin de la trame conditionnelle, de sorte que tout le texte qui suit sera compilé.

L'instruction `#else` doit toujours être celle qui précède le #endif.

Vous pouvez ajouter une autre condition à la trame conditionnelle :

```
#if condition1
    text A
#elif condition2
    text B
#endif
```

Dans le cas qui précède, si *condition1* est VRAIE, le texte A sera compilé, mais le texte B sera sauté et la compilation continuera à partir du `#endif`. Cependant, si *condition1* est FAUSSE, le texte A sera sauté pour se porter sur #elif lorsque *condition2* sera testée. Si alors *condition2* est VRAIE, le texte B sera compilé. "`#elif`" est identique à "`#elseif`".

L'ajout de `#else` à la trame conditionnelle ci-dessus donne :

```
#if condition1
    text A
#elif condition2
    text B
#else
    text C
#endif
```

Ici, si *condition1* est VRAIE, le texte A sera compilé, mais le texte B et le texte C seront ignorés et la compilation continuera à partir du `#endif`. A l'inverse, si *condition1* est FAUSSE, le texte A sera ignoré et on passera sur

#elif pour le test de *condition2*. Si alors *condition2* est VRAIE, le texte B sera compilé et le texte C sera ignoré; si, cependant, *condition2* est FAUSSE, le texte B sera ignoré et on passera sur `#else` et le texte C sera compilé.

Vous pouvez avoir autant de `#elif` (équivalent à `#elseif`) que vous le souhaitez dans chaque trame conditionnelle, mais il ne peut y avoir qu'un seul `else` par trame, et chaque `#if` doit obligatoirement avoir un `#endif` correspondant. Certains programmeurs n'hésitent pas à imbriquer les trames conditionnelles, mais cela peut devenir rapidement complexe. C'est une pratique de programmation à éviter autant que possible. A défaut, il est recommandé d'étiqueter soigneusement chaque `#endif` avec un commentaire afin de pouvoir voir à quel `#if` il se réfère.

Types de directive #if

#ifdef *expression*	où *expression* est un mot qui peut être défini dans le script de ressource ou dans un fichier d'en-tête. Cette instruction renvoie TRUE si l'expression est définie et FALSE si elle ne l'est pas. Le terme *expression* doit être un mot et non un nombre ni une chaîne entre guillemets.
#ifndef *expression*	comme ci-dessus, mais cette instruction renvoie FALSE si *expression* est définie et TRUE si elle ne l'est pas.
#if 0	cette instruction renvoie toujours FALSE et peut donc être utilisée pour sauter par-dessus un texte qui doit être ignoré.
#if *expression1 relational-operator expression2*	où, *expression1* doit être un mot défini ailleurs dans le fichier, dans un fichier d'inclusion ou sur la ligne de commande. Ce ne peut être un nombre. *relational operator* peut être l'un des opérateurs suivants : >= supérieur à ou égal <= inférieur à ou égal == égale = égale != différent de > supérieur à < inférieur à *expression2* peut être un nombre ou un mot défini ailleurs dans le fichier, dans un fichier d'inclusion ou sur la ligne de commande, qui évalue par rapport à un nombre. Par exemple, #if WINVER >= 400h retournera TRUE si WINVER est défini comme contenant la valeur 400h ou plus.
#if ! *expression*	ceci inverse le résultat de la condition, de telle sorte que `#if` n'est pas la même chose que #if! Par exemple, `# if! 0` retournera TRUE.

2.4 Autres Symboles

Il s'agit d'autres symboles qui peuvent être utilisés dans le script de ressources ou dans un fichier d'en-tête et qui ont une signification particulière pour le compilateur de ressources.

;	commentaire sur une ligne – tout ce qui se trouve après est ignoré jusqu'à la fin de la ligne
//	commentaire sur une ligne – tout ce qui se trouve après est ignoré jusqu'à la fin de la ligne
;; ... … ;;	bloc de commentaires – peut s'étendre sur plusieurs lignes et tout ce qui se trouve entre les marques de début et de fin est ignoré
/* ... ... */	bloc de commentaires – peut s'étendre sur plusieurs lignes et tout ce qui se trouve entre les marques de début et de fin est ignoré
\	symbole de continuation de ligne. Mis à la fin d'une ligne ou juste avant un commentaire, il permet au traitement de continuer avec le matériau sur la ligne suivante (ne peut pas être utilisé dans des chaînes de caractères entre guillemets ou des noms de fichiers)
{	même fonctionnalité que BEGIN
}	même fonctionnalité que END
–*number*	signifie que le nombre est négatif (par exemple -22h si un dword prend la valeur 0FFFFFFDEh)
~*number*	signifie que le nombre est inversé (par exemple, ~22h si un dword prend la valeur 0FFFFFFDDh)
+	signe plus pour l'addition (par exemple 22h+3h donne 25h)
-	signe moins pour la soustraction (par exemple 22h–3h donne 1Fh)
| ou ¦	OR bit à bit (par exemple, 22h | 3h produit le résultat 23h)

&	AND bit à bit (par exemple, 22h & 3h produit le résultat 2h)
!*valeur*	en arithmétique, cette symbolique provoque l'inversion de la *valeur*, qui est ensuite appliquée avec un AND à la *valeur* existante. Ceci revient au même que la transformation en une valeur négative
NOT *valeur*	– idem ci-dessus –
#	dans une macro #define, cette symbolique provoque l'ajout de guillemets au terme qui suit : - par exemple, #HELLO devient "HELLO" - par exemple, #(MAJOR.MINOR) devient "MAJOR.MINOR"
##	dans une macro `#define`, cette symbolique permet de joindre deux éléments, supprimant en même temps tous les espaces entre les deux par exemple, PATCH ## WORK devient PATCHWORK
(...)	dans une macro `#define`, le matériau entre parenthèses est évalué en premier

2.5 Nombres

Le compilateur de ressources reconnaît les types de nombres suivants, par exemple :

6666666h	nombre hexadécimal
3434343H	nombre hexadécimal
9999999D	nombre décimal
22553388d	nombre décimal
34567789	nombre décimal
0x456789	nombre hexadécimal
456L	spécifie que le nombre doit être enregistré comme un dword bien qu'ayant l'apparence d'un mot

2.6 Arithmétique

Le compilateur de ressources peut effectuer des opérations arithmétiques limitées sur les éléments suivants :

- identifiants, styles, flags ainsi que les positions et dimensions des dialogues et contrôles
- définition des directives
- expressions dans les directives conditionnelles

Exemples d'opérations arithmétiques valides

```
66h+2h CURSOR first.cur
#define BELLS 200h | 22h + 1h
BELLS+1000h ICON icon1.ico
BELLS-1h ICON icon1.ico
STYLE WS_TABSTOP | WS_GROUP | WS_DISABLED
DIALOG 0x34+2,0x20,0x300-0x60,400-200
```

Le compilateur de ressources ne peut pas résoudre des formules plus complexes impliquant par exemple des opérateurs logiques, des virgules décimales, des multiplications ou des divisions.

2.7 Chaînes et manipulations de chaînes

Les chaînes entre guillemets dans la ressource *stringtable* et dans tout le texte destiné aux dialogues et aux menus doivent être stockées en Unicode dans les fichiers de ressources binaires et dans l'EXE. Si le script de ressource est un fichier texte ANSI, cependant, les chaînes seront en caractères ANSI. Dans ce cas, le compilateur de ressources les convertit en Unicode à l'aide de l'API *MultiByteToWideChar*. Cette API utilise la page de codes courante pour cette conversion. Vous devez donc vous assurer qu'à la compilation, votre machine utilise la même page de code que celle utilisée lors de la création du script source.

Si le script de ressource est un fichier Unicode, les codespages ne sont pas utilisés; les chaînes sont utilisées telles qu'elles apparaissent dans le script de ressources.

Les chaînes sont placées dans le fichier RES (ou le fichier OBJ) en Unicode, que vous utilisiez ou non l'indicateur Unicode de style "C" L"*chaîne*". Le compilateur de ressources considère la forme L"*chaîne*" comme implicite pour ces types de chaînes. Il n'est donc pas nécessaire de l'utiliser dans votre script de ressources, sauf si vous souhaitez insérer des caractères Unicode spéciaux – voir ci-dessous.

Il n'y a pas d'indicateur L"*chaîne*" implicite pour les chaînes entre guillemets dans les ressources de données brutes (par exemple, RCDATA). Par conséquent, dans un fichier ANSI, ces chaînes ne sont pas converties en Unicode sauf si la forme L"*chaîne*" est spécifiée.

Toutes les chaînes entre guillemets peuvent contenir des séquences d'échappement traitées de manière spécifique par le compilateur de ressources :

&	dans les chaînes de titres de menu, ce symbole force le système, au démarrage, à créer un raccourci en direction de l'élément de menu souhaité utilisant le caractère suivant ce symbole. Si vous avez besoin d'une esperluette en tant que caractère, vous devez la spécifier 2 fois de suite, par exemple, &&
" "" "	double les guillemets pour les conserver. Par exemple """Hello there""" sera conservé comme "Hello there" et "He said ""Hello there"" quickly" sera conservé comme : He said "Hello there" quickly
""	chaîne vide (utile par exemple dans les contrôles statiques où le texte doit être inséré ultérieurement pendant l'exécution)
\t	tabulation (convertie en caractère TAB de code ASCII 9h)
\a	alerte (convertie en caractère BELL de code ASCII 8h) dans les menus, cette séquence est utilisée pour aligner tout le texte qui suit à droite
\n	nouvelle ligne (convertie en caractère LINEFEED de code ASCII 0Ah)
\r	retour-chariot (converti en caractère CARRIAGE RETURN de code ASCII 0Dh)
\\	converti en une barre oblique inverse (backslash) uniquement
nombre	voir ci-dessous
autre	la barre oblique inverse est considérée comme un backslash ordinaire

2.7.1 Séquence *nombre* permettant de produire un caractère spécial

Dans toutes les chaînes entre guillemets qui sont converties par le compilateur de ressources, vous pouvez rencontrer la séquence :

nombre

Si tel est le cas, elle provoquera alors l'insertion dans le texte du caractère dont le code est donné par le paramètre *nombre*. Par exemple, "Copyright \251 2005" insère un symbole © dans le texte. Étrange ! Le symbole ©, comme chacun sait, possède, en ANSI, la valeur 169, ou A9 en hexa.

Aussi étonnant que cela puisse paraître, le compilateur de ressources utilise le système octal pour représenter ces codes plutôt que le système décimal. Nous passons donc dans un système en base 8, c'est-à-dire un système dans lequel les chiffres ne peuvent dépasser 7. Pourquoi ? C'est simplement pour assurer la compatibilité avec les scripts de ressources existants.

Si l'on reprend l'exemple ci-dessus, un calcul rapide montre en effet que 251 en octal correspond à 169 en décimal et à A9 en hexa.

Quelquefois, *nombre* peut paraître plutôt étrange. Par exemple "The entrance fee was \4423", est converti en "The entrance fee was $23". Ici, le compilateur de ressources n'essaie pas de trouver le caractère 442 octal ou 4423 octal car il sait d'emblée que cette valeur est trop élevée (256 en décimal correspond à 400 en octal). Cela fonctionne bien pour les caractères au-dessus de 37 octal, mais pour les caractères de valeur inférieure, par exemple 12 octal qui correspond à la valeur 0Ah (saut de ligne), il pourrait y avoir confusion. Par exemple, la phrase "On a trouvé en moyenne environ\1223 mouches dans chacune des soupes" devrait être écrite "Nous avons trouvé en moyenne environ\01223 mouches dans chacune des soupes". On le voit, il est difficile de faire ressortir clairement cette moyenne de 23 mouches… Cela est dû au fait que le compilateur de ressources ne prend en considération que 3 chiffres en octal et pas un de plus. Donc il s'arrête après le \012 (saut de ligne) et la valeur 23 demeure dans le texte. Bien sûr, dans le cas présent, vous pourriez préférer l'utilisation de \n qui provoque un Line Feed sans ambigüité (voir tableau au début du paragraphe 2.7).

2.7.2 Séquence *nombre* en hexa plutôt qu'en octal

Vous pouvez également utiliser des valeurs hexadécimales lors de la saisie de *nombre*, en incluant *x* après la barre oblique inverse. Par exemple, la version hexadécimale "Copyright \xA9 2005" donne le même résultat que "Copyright \251 2005" qui est la variante octale. De nouveau, le compilateur de ressources ne regardera pas au-delà du deuxième chiffre hexadécimal pour la valeur. Alors assurez-vous que, lorsque de petites valeurs sont utilisées, un zéro soit inséré s'il y a risque de confusion. Par exemple, dans la phrase "Le condensateur a une valeur de 3 \xb5farads" (censée exprimer la valeur de "3 µfarads"), le compilateur de ressources ne prend pas en considération le nombre hexadécimal b5f, car de valeur trop élévée, mais se limite à la valeur hexa b5 qui caractérise le symbole µ. Cependant, dans la chaîne "Il est retourné sur Terre \x0aaprès 44 ans avec 5 personnes de plus à bord", le zéro est nécessaire pour éviter d'utiliser le nombre hexadécimal aa comme caractère spécial.

2.7.3 Séquence *nombre* en chaînes Unicode

Le compilateur de ressources recherche jusqu'à quatre caractères hexadécimaux dans *nombre* si la chaîne est explicitement déclarée comme chaîne Unicode. Vous pouvez créer cette situation en utilisant l'indicateur de style C L"*chaîne*" pour décrire ladite chaîne. Par exemple, pour créer une chaîne pour un bouton de dialogue affichant un Shekel (monnaie israëlienne), utilisez la chaîne suivante :

```
L"\x20AA"
```

Cela crée le caractère spécial Unicode 20AAh (le symbole du Shekel).

Vous n'auriez pas besoin de le faire si vous utilisiez des fichiers au format Unicode, car vous pourriez afficher le Shekel normalement et il serait codé correctement par le compilateur de ressources.

2.8 Le jeu de caractères Windows

Le jeu de caractères utilisé lorsque la chaîne est affichée par le menu, le dialogue ou toute autre méthode, dépend du jeu de caractères utilisé dans l'ordinateur exécutant l'EXE. Pour les ordinateurs anglo-saxons, il s'agit généralement du jeu de caractères Windows. Les table de valeurs octales n'étant pas d'un usage répandu, vous en trouverez une ci-après, immédiatement suivie par son équivalente en valeurs hexadécimales.

2.8.1 Codes ASCII du jeu de caractères Windows en *octal*

Space	!	"	#	$	%	&	'	(	)	*	+	,	-
40	41	42	43	44	45	46	47	50	51	52	53	54	55
.	/	0	1	2	3	4	5	6	7	8	9	:	;
56	57	60	61	62	63	64	65	66	67	70	71	72	73
<	=	>	?	@	A	B	C	D	E	F	G	H	I
74	75	76	77	100	101	102	103	104	105	106	107	110	111
J	K	L	M	N	O	P	Q	R	S	T	U	V	W
112	113	114	115	116	117	120	121	122	123	124	125	126	127
X	Y	Z	[	\	]	^	_	`	a	b	c	d	e
130	131	132	133	134	135	136	137	140	141	142	143	144	145
f	g	h	i	j	k	l	m	n	o	p	q	r	s
146	147	150	151	152	153	154	155	156	157	160	161	162	163
t	u	v	w	x	y	z	{	|	}	~	□		
164	165	166	167	170	171	172	173	174	175	176	177	200	201
,	*f*	„	…	†	‡	ˆ	‰	Š	‹	Œ			
202	203	204	205	206	207	210	211	212	213	214	215	216	217
	‘	’	“	”	•	–	—	˜	™	š	›	œ	
220	221	222	223	224	225	226	227	230	231	232	233	234	235
	Ÿ		¡	¢	£	¤	¥	¦	§	¨	©	ª	«
236	237	240	241	242	243	244	245	246	247	250	251	252	253
¬		®	¯	°	±	²	³	´	µ	¶	·	¸	¹
254	255	256	257	260	261	262	263	264	265	266	267	270	271
º	»	¼	½	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
272	273	274	275	276	277	300	301	302	303	304	305	306	307
È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ
310	311	312	313	314	315	316	317	320	321	322	323	324	325
Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã
326	327	330	331	332	333	334	335	336	337	340	341	342	343
ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ
344	345	346	347	350	351	352	353	354	355	356	357	360	361
ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
362	363	364	365	366	367	370	371	372	373	374	375	376	377

2.8.2 Codes ASCII du jeu de caractères Windows en *hexadécimal*

Space	!	"	#	$	%	&	'	(	)	*	+	,	-
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D

.	/	0	1	2	3	4	5	6	7	8	9	:	;
2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B
<	=	>	?	@	A	B	C	D	E	F	G	H	I
3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49
J	K	L	M	N	O	P	Q	R	S	T	U	V	W
4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57
X	Y	Z	[	\	]	^	_	`	a	b	c	d	e
58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65
f	g	h	i	j	k	l	m	n	o	p	q	r	s
66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73
t	u	v	w	x	y	z	{	|	}	~	□		
74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81
,	*f*	„	…	†	‡	ˆ	‰	Š	‹	Œ			
82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	‘	’	“	”	•	–	—	˜	™	š	›	œ	
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D
	Ÿ		¡	¢	£	¤	¥	¦	§	¨	©	ª	«
9E	9F	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB
¬		®	¯	°	±	²	³	´	µ	¶	·	¸	¹
AC	AD	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9
º	»	¼	½	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
BA	BB	BC	BD	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7
È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ	Ò	Ó	Ô	Õ
C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3	D4	D5
Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	à	á	â	ã
D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3
ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	ð	ñ
E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF	F0	F1
ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ
F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

2.9 Instructions des Ressources

Les instructions des ressources sont celles du Compilateur de Ressources. Elles contiennent les informations à partir desquelles le fichier de sortie est réellement créé et se divisent en quatre catégories principales :

- Types de ressources de données brutes prenant les données d'un fichier ou d'un script de ressources
- La ressource *stringtable*
- Les types de ressources d'informations
- Ressources pour l'interface utilisateur

2.9.1 Types de ressources de données brutes prenant les données à partir d'un fichier

Avec ces ressources, le compilateur de ressources prend simplement les données brutes d'un fichier d'entrée et les ajoute au fichier de sortie (parfois avec un traitement supplémentaire). Par conséquent, les données de ces fichiers se dirigent vers l'EXE, au moment de l'édition de liens, pour être utilisé pendant l'exécution. C'est un moyen pratique de conserver les données de ces fichiers dans le fichier EXE. Une alternative serait de juxtaposer les fichiers individuels au produit final pour qu'ils soient chargés à l'exécution, mais le risque de les perdre lors d'un quelconque transfert doit néanmoins être pris en considération.

Les données (en tant que données brutes) peuvent être utilisées à l'exécution en appelant les API *FindResource* et *LoadResource*, mais certaines API spécifiques permettent également de manipuler les données. Les différents types de ressources de données brutes qui fonctionnent en utilisant des fichiers, leurs numéros de type de ressource et l'API spécifique (le cas échéant) pour les charger pendant l'exécution sont :

CURSOR	1	*LoadCursor* ou *LoadImage*
BITMAP	2	*LoadBitmap* ou *LoadImage*
ICON	3	*LoadIcon* ou *LoadImage*

FONT	8	*n'est plus utililisé*
RCDATA	10	
MESSAGETABLE	11	FormatMessage
PLUGPLAY	19	
VXD	20	
ANICURSOR	21	
ANIICON	22	
HTML	23	
MANIFEST	24	
User-defined resource		

La *ressource définie par l'utilisateur* (*User defined resource*) est une ressource contenant uniquement des données brutes, qui sont du *type* défini par l'utilisateur. Elle peut être définie par son nom, par exemple, MyRes, ou par un numéro qui peut être n'importe quel nombre de 16 bits au-dessus de 0FFh. Les valeurs 1 à 0FFh sont réservées en effet aux ressources définies par le système.

La syntaxe générale de ce type de ressources dans le script de ressources (utilisant CURSOR, par exemple) est la suivante :

nameID CURSOR *filename*

Où,

nameID peut être, soit :
- un nombre 16 bits allant de 1 à 7FFFh
- une expression qui évalue ce nombre
- un nom (à l'exception de FONT, une ressource FONT ne pouvant être identifiée par un nom)

filename peut être, soit :
- une chaîne encadrée de guillemets
- une chaîne sans guillemets
- une chaîne encadrée par les symboles < et >

Sauf si le fichier est dans le repertoire courant, le chemin d'accès complet doit être indiqué.

Exemples

```
2 CURSOR happy.cur
0x6666 BITMAP c:\bitmaps\happy.bmp
MyIcon ICON "happy.ico"
#define ID_MINE 22
ID_MINE ICON <happier.ico>
44h RCDATA happy.txt
ID_MINE RCDATA happier.txt
1 MANIFEST "YourApp.manifest"    ; (ressource définie par l'utilisateur)
788h MYRES happier.txt           ; (ressource définie par l'utilisateur)
788h 100h happier.txt            ; (ressource définie par l'utilisateur)
```

2.9.2 Types de ressources de données brutes prenant les données à partir du script de ressources

Ces ressources vous permettent d'insérer des données brutes dans un EXE via le fichier de ressource binaire. Le compilateur de ressources ne modifie pas les données brutes. À moins d'utiliser la capacité du système à distinguer les ressources de différentes langues, les programmeurs en assembleurs ne trouvent pas nécessairement grand intérêt à cette pratique tant il est plus simple d'utiliser la section de données pour récupérer ces informations.

Les données (en tant que données brutes) peuvent être utilisées à l'exécution en appelant les API *FindResource* et *LoadResource*.

Ce type de ressource correspond à RCDATA (type numéro 10) lorsqu'il est utilisé avec BEGIN et END, et non avec un nom de fichier. Vous pouvez également choisir n'importe quel nom pour la ressource, auquel cas il fonctionne de la même manière que RCDATA, sauf que le type de ressource a un nom. C'est ce qu'on appelle une *ressource définie par l'utilisateur*. Vous pouvez également simplement fournir un nombre de 16 bits au-dessus de 0FFh pour le type de ressource.

La syntaxe de RCDATA utilisée de cette manière est la suivante :

nameID RCDATA
[*définition des instructions*]
BEGIN

```
    données brutes
END
```

Ou, dans le cas d'une *ressource définie par l'utilisateur* :

```
nameID type
[définition des instructions]
BEGIN
    données brutes
END
```

où,

nameID peut être, soit :
- un nombre 16 bits allant de 1 à 7FFFh
- une expression qui évalue ce nombre
- un nom

définition-instructions présence non indispensable, mais peuvent être, soit :
- une instruction VERSION
- une instruction CHARACTERISTICS
- une instruction LANGUAGE

voir **ci-après** pour les détails

type peut être, soit :
- un nombre 16 bits supérieur à 0FFh
- une expression dont le résultat est un nombre 16 bits supérieur à 255
- un nom en caractères

données brutes peuvent être, soit :
- une chaîne ANSI encadrée de guillemets
- une chaîne Unicode encadrée de guillemets commençant par L"
- une valeur numérique stockée dans un word
- une valeur numérique stockée dans un dword (se terminant par L)

Exemples

```
0x3333 RCDATA
BEGIN
    "Hello world"
    "Hello world (zero terminated)\0"
    L"A Unicode version of the above\0"
    0x9999      ; nombre hexadécimal stocké dans un word
END

MyRes RCDATA
BEGIN
    1034        ; nombre décimal stocké dans un word
END

MyRes MyResType
BEGIN
    10456L                  ; nombre décimal stocké dans un dword
    1234L,56666L,99999L     ; nombres décimaux stockés dans des dwords
END

34h 100h
BEGIN
    33hL,34hL,35hL,36hL     ; nombres hexadécimaux stockés dans des dwords
    0x37L,0x38L,0x39L,0x40L ; nombres hexadécimaux stockés dans des dwords dans le style C
END
```

2.9.3 La Ressource STRINGTABLE

La ressource STRINGTABLE permet d'entreposer des chaînes dans le fichier EXE, récupérables, lorsque cela est nécessaire, au moment de l'exécution. Pour les programmeurs en assembleur, cela constitue une alternative au

libellé des chaînes dans le segment de données. Même si vous devez utiliser des chaînes Unicode et que vous utilisez GoAsm comme assembleur, il est généralement plus facile de conserver les chaînes dans les données plutôt que d'utiliser la ressource STRINGTABLE.

Si le fichier source est au format ANSI, les chaînes de la ressource STRINGTABLE sont automatiquement converties en Unicode par le compilateur de ressources. Il n'est donc pas nécessaire d'utiliser l'indicateur Unicode de style C L"*chaîne*". Le compilateur de ressources considère l'indicateur L"*chaîne*" comme implicite. La conversion est effectuée à l'aide de l'API *MultiByteToWideChar* qui utilise la page de codes courante à cet effet. Vous devez donc vous assurer qu'à la compilation, votre machine utilise la même page de codes que celle utilisée lors de la création du script source.

Si le fichier source est au format Unicode, aucune conversion n'a lieu et les pages de code ne sont pas utilisées ; les chaînes sont utilisées telles qu'elles apparaissent dans le script de ressources.

L'une des raisons pour lesquelles le programmeur en assembleur pourrait souhaiter entreposer des chaînes dans une ressource STRINGTABLE plutôt que dans le segment de données serait de pouvoir tirer parti de la capacité du système à distinguer les chaînes de différentes langues au moment de l'exécution.

Les chaînes sont conservées par blocs de 16 unités. Chaque bloc aura la même langue et les mêmes 12 bits supérieurs de l'ID qui ont été spécifiés dans le script de ressource. Les 4 bits les moins significatifs sont masqués et ne sont pas enregistrés car il est supposé que les ID s'exécutent en séquence dans chaque bloc de 0 à 0Fh. Pour que cette hypothèse soit valide, si les ID de chaîne ne sont pas séquentiels, le compilateur de ressources doit insérer des chaînes vides. Pour réduire l'espace requis, il est donc préférable de privilégier la séquentialité des identifiants.

Pour récupérer l'une des chaînes, appelez *LoadString* en fournissant son identifiant. Elle sera copiée ensuite dans un tampon avec un terminateur nul.

La syntaxe de la ressource STRINGTABLE est la suivante :

```
STRINGTABLE
[définition des instructions]
BEGIN
  stringID "string"
  .....
END
```

Où,

stringID	peut être : - un nombre 16 bits qui identifie la chaîne - une expression dont le résultat aboutit à ce nombre
"string"	peut être : - une chaîne entre guillemets n'excédant pas 4097 caractères - une expression qui est définie ailleurs sous la forme d'une chaîne

Si la chaîne s'étend au-delà d'une ligne, le compilateur de ressources insère un espace et un caractère de saut de ligne (ASCII 0Ah) au saut de ligne et tous les espaces en tête de la ligne suivante de votre script de ressource sont supprimés.

2.9.4 Types de ressources d'information

Ces types de ressources sont utilisés pour conserver des informations sur l'EXE dans un certain format, pour l'utilisation par des outils ou pour des informations à destination de l'utilisateur.

Il existe deux types de ces ressources.

Tout d'abord DLGINCLUDE (type de ressource 17) qui contient un nom de fichier lisible par l'éditeur Microsoft Dialog Editor. Ce nom de fichier est celui du fichier include contenant des définitions lors de la création de boîtes de dialogue. Cette information n'est pas réellement conservée dans le fichier EXE et est ignorée par l'éditeur de liens.

Sa syntaxe est :

nameID DLGINCLUDE *filename*

Où,

nameID peut être, soit :
- un nombre 16 bits de valeur 1 à 7FFFh
- une expression qui évalue ce nombre
- un nom

filename peut être, soit :
- une chaîne entre guillemets
- une chaîne non encadrée de guillemets
- une chaîne encadrée par les caractères < et >
- sauf si le fichier se trouve dans le répertoire courant, le chemin complet doit être spécifié

2.9.5 La ressource VERSIONINFO

La ressource VERSIONINFO (type de ressource 16) est utilisée pour conserver des informations sur la version de l'EXE. Elle peut être nécessaire lors de son installation ou en cas de mise à jour.

Les informations de version peuvent être récupérées de différentes manières à l'aide des API *GetFileVersionInfo*, *GetFileVersionInfoSize* et *VerQueryValue*. C'est aussi l'information qui apparaît dans la feuille de propriétés "Version" lorsque vous consultez les "propriétés" d'un fichier exécutable dans Windows.

La syntaxe de cette ressource est :

```
1 VERSIONINFO
[fixed-info statements]
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK lang-charset
        BEGIN
            string-value statements
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation"
        lang/char statements
    END
END
```

Où,

instructions fixed-info peut être l'une ou plusieurs des instructions suivantes avec des données comme dans les exemples suivants. Les données sont utilisées par le compilateur de ressources pour remplir une structure de 13 dwords qui est conservée dans le fichier de ressources binaires et donc dans le fichier EXE (la structure VS_FIXEDFILEINFO)

FILEVERSION 3h,0Ah,0,3Dh
cette valeur doit être écrite comme quatre valeurs de 16 bits qui sont en fait enregistrées comme deux dwords : dans cet exemple, le premier dword (le plus significatif) est 3000Ah et le second dword (le moins significatif) est 3Dh, produisant une valeur de version égale à 3000A 0000003Dh.

PRODUCTVERSION 6666h,0000h,0000h,7777h
définit une version de produit de valeur 66660000 00007777h

FILEFLAGSMASK 3Fh désigne les bits du paramètre suivant qui sont valides

FILEFLAGS *fileflags* qui peut être un ou plusieurs des flags ou valeurs suivants :

VS_FF_DEBUG	(1= le fichier contient des informations de débogage)
VS_FF_PATCHED	(4= l'information de version a été changée par rapport à la version d'expédition originale)
VS_FF_PRERELEASE	(2= le fichier est une version de développement seulement et n'est pas

disponible dans le commerce)

VS_FF_PRIVATEBUILD (8= le fichier n'a pas été élaboré en utilisant les procédures de publication standard)

VS_FF_SPECIALBUILD (20h= le fichier a été élaboré en utilisant des procédures de publication standard mais c'est une variation du fichier standard du même numéro de version)

FILEOS *fileos* où *fileos* spécifie le système d'exploitation pour lequel le fichier a été conçu et peut prendre l'une des valeurs suivantes :

VOS_UNKNOWN (valeur zéro - inconnu)
VOS_DOS (1000h = fichier conçu pour DOS)
VOS_NT (4000h = fichier conçu pour Windows NT)
VOS__WINDOWS32 (4 = fichier conçu pour Windows 32 bits)
VOS_NT_WINDOWS32 (40004h= fichier conçu pour Windows 32 bits fonctionnant sous Windows NT)

FILETYPE *filetype* où *filetype* spécifie le type général du fichier et peut prendre l'une des valeurs suivantes :

VFT_UNKNOWN (valeur zéro - inconnu)
VFT_APP (1= c'est une application)
VFT_DLL (2= c'est une DLL)
VFT_DRV (3= c'est un pilote de périphérique)
VFT_FONT (4= c'est une police de caractères)
VFT_VXD (5= c'est un périphérique virtuel)
VFT_STATIC_LIB (7= c'est une bibliothèque de lien statique)

FILESUBTYPE *subtype* où *subtype* spécifie plus d'informations sur le type de fichier et, si *filetype* est égal à VFT_DRV, peut prendre l'une des valeurs suivantes :

VFT2_UNKNOWN (valeur zéro - inconnu)
VFT2_PRINTER (1= c'est un pilote d'imprimante)
VFT2_KEYBOARD (2= c'est un pilote de clavier)
VFT2_LANGUAGE (3= c'est un pilote de langage)
VFT2_DISPLAY (4= c'est un pilote d'affichage)
VFT2_MOUSE (5= c'est un pilote de souris)
VFT2_NETWORK (6= c'est un pilote de réseau)
VFT2_SYSTEM (7= c'est un pilote système)
VFT2_DRV_INSTALLABLE (8= c'est un pilote installable)
VFT2_DRV_SOUND (9= c'est un pilote audio)
VFT2_DRV_COMM (0Ah= c'est un pilote de communications)

si *filetype* est égal à VFT_FONT, il peut prendre l'une des valeurs suivantes.

VFT2_FONT_RASTER (1= police bitmap)
VFT2_FONT_VECTOR (2= police vectorielle)
VFT2_FONT_TRUETYPE (3= police TrueType)

si *filetype* est égal à VFT_VXD, cette valeur doit être l'identifiant du périphérique virtuel contenu dans le bloc de contrôle de périphérique virtuel.

Les noms des *instructions fixed-info* sont fixes et sont utilisés par le compilateur de ressources pour décider où placer la valeur spécifiée dans la structure VS_FIXEDFILEINFO. Comme vous pouvez le voir, certaines valeurs ont des significations définies réservées par Microsoft, mais il n'y a aucune raison pour que vous ne puissiez pas utiliser vos propres valeurs qui peuvent avoir une signification particulière pour votre application.

lang-charset est une chaîne entre guillemets contenant (en chiffres) un nombre hexadécimal qui identifie la langue et le jeu de caractères utilisés dans les *instructions string-value* qui suivent. La valeur la plus fréquemment trouvée dans ce texte est "040904E4" qui signifie US English + Windows, jeu de caractères multilingue.

Les instructions *lang/char* consistent en deux nombres de 16 bits séparés par une virgule. Le premier identifie une langue et le second, un jeu de caractères supportés par l'application. Si une instruction *lang/char* est incluse dans la ressource VERSIONINFO, le système initialise la feuille de propriétés "Version" trouvée lors de l'examen des "Propriétés" du fichier exécutable et inclut une clé "Langue". La valeur la plus commune trouvée est 0x409,1252 qui est US English + Windows, jeu de caractères multilingue, avec ce dernier étant donné dans sa valeur déci-

	male, mais 4E4h conviendrait aussi bien.
instructions string-value	série d'instructions contenant deux chaînes entre guillemets avec la syntaxe suivante : *"name", "value"*
"*name"*	Il s'agit d'une chaîne qui sera donnée à l'API *VerQueryValue* afin de récupérer *value*. Les chaînes suivantes sont recherchées par le système lors de l'affichage de la feuille de propriétés "Version" : "FileVersion", "FileDescription" et "LegalCopyright", et *value* apparaît en haut de la feuille de propriétés adjacente aux mots "File Version", "Description" et "Copyright", respectivement. D'autres chaînes apparaissent dans les listbox dans la feuille de propriétés. Vous pouvez utiliser la chaîne de votre choix, mais si elles sont trop longues, elles sont tronquées dans la feuille de propriétés et apparaissent toujours dans l'ordre alphabétique. Certaines des chaînes suivantes sont souvent utilisées : "Comments" "CompanyName" "InternalName" "LegalTradmarks" "OriginalFilename" "PrivateBuild" "ProductName" "ProductVersion" "SpecialBuild"
"*value*"	est une valeur définie par l'utilisateur dans une chaîne entre guillemets qui correspond à l'une des chaînes standard et sera renvoyée par *VerQueryValue*.

Exemples de ressources VERSIONINFO

```
1  VERSIONINFO
FILEVERSION            2,0,0,0
PRODUCTVERSION   2,0,0,0
FILEFLAGSMASK     0x0000003FL
FILEFLAGS              0x0000000BL
FILEOS                   0x00010001L
FILETYPE                0x00000001L
FILESUBTYPE          0x00000000L
BEGIN
      BLOCK "StringFileInfo"
      BEGIN
            BLOCK "040904E4"
            BEGIN
                  VALUE "CompanyName", "MySoftware Company"
                  VALUE "Contact e-mail", "MySoftware@ether.com"
                  VALUE "FileDescription","MySoftware.Exe"
                  VALUE "FileVersion", "3.45"
                  VALUE "DevelopmentFile","Dev345.asm"
                  VALUE "LegalCopyright","Copyright\251 2005 Co"
                  VALUE "Resources made using","GoRC.Exe"
            END
      END
      BLOCK "VarFileInfo"
      BEGIN
            VALUE "Translation", 0x0409,1252
      END
END
```

2.9.6 Ressources destinées à l'interface utilisateur

Ces types de ressources facilitent la création de raccourcis, de menus et de boîtes de dialogue, les trois ingrédients de base de toute interface utilisateur. Dans chaque cas, il est toutefois possible d'utiliser les API en lieu et place de fichiers de ressources, mais c'est moins pratique et on ne peut pas tirer avantage du fait que le système puisse discriminer automatiquement les menus et les dialogues de différentes langues.

Pour utiliser ces ressources dans une application en langue unique, une API spécifique est appelée afin d'obtenir un handle vers la ressource.

Les cinq types de ressources concernées, les numéros de type de ressource ainsi que les API utilisées pour charger les ressources sont :

ACCELERATORS	9	LoadAccelerators
MENU	4	LoadMenu
MENUEX	4	LoadMenu
DIALOG	5	CreateDialogParam, DialogBoxParam
DIALOGEX	5	CreateDialogParam, DialogBoxParam

2.10 Les ressources de raccourci clavier (ACCELERATORS)

La syntaxe pour la resource ACCELERATORS se présente comme suit :

nameID ACCELERATORS
[*defining-statements*]
BEGIN
    event, result [,*type/options*]
    .....
END

Où,

nameID — peut être, soit :
- un nombre 16 bits allant de 1 à 7FFFh
- une expression qui évalue ce nombre
- un nom

définition-instructions — présence non indispensable, mais peuvent être, soit :
- une instruction VERSION
- une instruction CHARACTERISTICS
- une instruction LANGUAGE

voir **ci-après** pour les détals

event — spécifie la frappe d'une touche ou d'une séquence de touches qui va entraîner l'envoi du résultat à la boucle de message du process qui a chargé les raccourcis. Cela peut être :
- un simple caractère entre guillemets
  ex. "a" signifie la touche "a"
- un simple caractère précédé d'un caret
  ex. "^A" caractérise une séquence Crtl key
  si ^ est utilisé, le caractère doit être une lettre majuscule
- un nombre 16 bits
- une expression dont le résultat est un nombre 16 bits
  ex. les VK_names couramment utilisés dans Winuser.h

result — peut être, soit :
- un nombre 16 bits
- une expression dont le résultat est un nombre 16 bits

type/options — peut être l'une des valeurs suivantes, séparées par des virgules :

ASCII	(valeur zéro, ce qui signifie que la valeur de l'événement fait référence à une valeur de caractère ASCII)
VIRTKEY	(1 = signifie que la valeur de l'événement fait référence à une valeur de touche virtuelle.

Si ni ASCII, ni VIRTKEY ne sont spécifiés, alors ASCII est implicite)

SHIFT	(4 = l'événement ne se produit que si la touche Maj est enfoncée)
CONTROL	(8 = l'événement ne se produit que si la touche Ctrl est enfoncée)
ALT	(10h = l'événement ne se produit que si vous appuyez sur la touche Alt)
NOINVERT	(2 = ne mettez pas en surbrillance un élément de menu de niveau supérieur si possible lorsqu'un accélérateur est utilisé)

Exemples de ressources ACCELERATORS (Raccourcis)

```
0x20 ACCELERATORS
BEGIN
        "N", 20h, CONTROL
        "^A", 0x1111
        "K", 0x2222
        "k", 0x3333, ALT
        98, 0x4444, ASCII
        66, 0x5555, ASCII
        "g", 0x6666
        "G", 0x7777
        "K", 0x2222, VIRTKEY
        67, 0x5556, VIRTKEY
        68, 0x5557, VIRTKEY, NOINVERT
        VK_F1, 0x8888, VIRTKEY
        VK_F2, 0x8889, CONTROL, VIRTKEY
        VK_F3, 0x888A, SHIFT, VIRTKEY
        VK_LBUTTON, 0x888B, ALT, VIRTKEY
        VK_NUMPAD4, 0x888C, ALT, SHIFT,VIRTKEY
        VK_SPACE, 0x888D, CONTROL, SHIFT,VIRTKEY
        VK_END, 0x888E, ALT, CONTROL,VIRTKEY
END
```

2.11 La ressource MENU

La syntaxe pour la ressource MENU se présente comme suit :

```
nameID MENU
[defining-statements]
BEGIN
        MENUITEM "text", id [,type/state]
        POPUP "text" [,type/state]
        BEGIN
                MENUITEM "text", id [,type/state]
        END
        MENUITEM SEPARATOR
        .........
END
```

La syntaxe pour la ressource MENUEX se présente comme suit :

```
nameID MENUEX [,helpID]
[defining-statements]
BEGIN
        MENUITEM "text", id [,type/state]
        POPUP "text", [id] [,type/state][,helpID]
        BEGIN
                MENUITEM "text", id [,type/state]
        END
        MENUITEM SEPARATOR
END
```

Où,

nameID	peut être, soit : - un nombre 16 bits allant de 1 à 7FFFh - une expression dont le résultat est ce nombre 16 bits - un nom
defining-statements	présence non indispensable, mais peuvent être, soit : - une instruction VERSION - une instruction CHARACTERISTICS - une instruction LANGUAGE voir **ci-après** pour les détails

"text" est une chaîne entre guillemets contenant le texte du menu ou de l'élément popup, qui peut également contenir :

- \t qui insère une tabulation
- \a qui aligne à droite tout le texte qui le suit
- & le système utilise le caractère suivant comme une touche de raccourci
- && utilise deux esperluètes pour en écrire une seule

Si un élément de menu est constitué simplement par 2 guillemets sans aucun texte, cela produit le même résultat qu'un SEPARATOR. Si vous voulez une ligne de menu complètement vide, utilisez plutôt un espace entre guillemets.

id est la valeur qui est envoyée à la procédure de fenêtre de la fenêtre qui héberge le menu. Cela peut être :

- un nombre 16 bits (MENU) ou un nombre 32 bits (MENUEX)
- une expression dont le résultat est un des nombres ci-dessus

type/state est un ou plusieurs des mots suivants séparés par des espaces ou l'opérateur OR | (barre verticale).

type

MENUBREAK 40h
met l'élément de menu suivant sur une nouvelle ligne ou dans une nouvelle colonne dans le cas de menus contextuels (popup)

MENUBARBREAK 20h
comme ci-dessus, mais comprend une ligne de séparation

OWNERDRAW 100h
le process est responsable du dessin du menu

RADIOCHECK 200h
utilisation d'un bouton radio à la place de la coche pour les éléments cochés

RIGHTJUSTIFY 4000h
justification à droite du texte du menu uniquement, pas sur les popups

HELP 4000h
identifie un élément d'aide

SEPARATOR 800h
pour les popups seulement

state

CHECKED 8h
vérifie l'élément de menu

GRAYED 3h
désactive et met en grisé l'élément de menu

DISABLED 3h
comme ci-dessus

INACTIVE 2h
élément de menu non grisé mais ne pouvant être sélectionné

HILITE 80h
Mise en surbrillance de l'élément de menu

DEFAULT 1000h
affiche le texte en gras et fait en sorte que la touche Entrée agisse de la même manière qu'un clic de souris sur cet élément

Au lieu d'utiliser les mots *type/state*, vous pouvez utiliser leur valeur respective en lieu et place, mais assurez-vous qu'ils sont insérés comme suit :

type,state
ou *type+type,state+state*

(les valeurs de type sont en premier, puis vient ensuite une virgule, puis enfin les valeurs d'état)

helpID (MENUEX uniquement) est la valeur envoyée à la procédure de fenêtre de la fenêtre qui héberge le menu si vous appuyez sur F1 lorsque le menu est actif. Cela peut être :

- un nombre 32 bits
- une expression dont le résultat est un nombre 32 bits

Important : dans l'instruction POPUP, pour garantir que le *helpID* est correcte-

	ment identifié, assurez-vous qu'il existe au total 3 virgules entre *id* et *helpID*. Cela indique au compilateur de ressources que *helpID* n'est pas un *type,* ni une valeur de *style*.
MENUITEM SEPARATOR	crée une ligne horizontale dans le menu qui est utile pour séparer les différents types d'éléments de menu

Exemples de ressources MENU

```
#define IDM_NEW 10h
#define IDM_OPEN 20h
GENERIC MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New...", IDM_NEW, GRAYED
        MENUITEM "&Open...", IDM_OPEN, GRAYED
        MENUITEM SEPARATOR
        POPUP "Send to"
        BEGIN
            MENUITEM "Drive a:"
        END
    END
    POPUP "View"
    BEGIN
        MENUITEM "Toolbar\tCrtl+T", 33h, CHECKED | HILITE
    END
END
```

Exemples de ressources MENUEX

```
#define IDM_NEW 10h
#define IDM_OPEN 20h
GENERIC MENUEX 99999999h
BEGIN
    POPUP "nice day", 0x100, CHECKED,,6666666h
    BEGIN
        MENUITEM "&New...", IDM_NEW, MENUBREAK | GRAYED
        MENUITEM "&Open...", IDM_OPEN, GRAYED
        MENUITEM SEPARATOR
        POPUP "Send to"
        BEGIN
            MENUITEM "Drive a:"
        END
    END
    POPUP "View"
    BEGIN
        MENUITEM "Toolbar", 33h, CHECKED | HILITE
    END
END
```

2.12 La ressource DIALOG

La synstaxe générale pour la ressource DIALOG est :

```
nameID DIALOG x, y, width, height
[defining-statements]
[dialog-statements]
BEGIN
    iconcontrol
    textcontrol
    non-textcontrol
    generic-control
END
```

La synstaxe générale pour la ressource DIALOGEX est :

nameID DIALOGEX *x, y, width, height* [,*helpID*]
[*defining-statements*]
[*dialog-statements*]
BEGIN
    iconcontrol
    textcontrol [,*helpID*]
    non-textcontrol [,*helpID*]
    generic-control [,*helpID*]
END

Où,

nameID — peut être, soit :
- un nombre 16 bits allant de 1 à 7FFFh
- une expression dont le résultat est ce nombre 16 bits
- un nom

x — est la position horizontale du dialogue exprimée en unités de dialogue

y — est la position verticale du dialogue exprimée en unités de dialogue

width — est la largeur du dialogue exprimée en unités de dialogue

height — est la hauteur du dialogue exprimée en unités de dialogue

helpID — (sur DIALOGEX uniquement) est la valeur envoyée à la procédure de dialogue si vous appuyez sur F1 lorsque la boîte de dialogue est active. Un *helpID* spécifié pour un contrôle est envoyé si l'utilisateur clique sur le contrôle après avoir déplacé le point d'interrogation d'aide. *helpID* peut être :
- un nombre 32 bits
- une expression dont le résultat est un nombre 32 bits

définition-instructions — présence non indispensable, mais peuvent être, soit :
- une instruction VERSION
- une instruction CHARACTERISTICS
- une instruction LANGUAGE

voir **ci-après** pour les détails

dialog-statements — présence non indispensable, mais peuvent être une ou plusieurs des instructions qui suivent :

CAPTION "*text*" — où "texte" est le titre du dialogue à insérer dans la barre de titre

CLASS *class* — Cette instruction est utilisée si vous avez enregistré votre propre classe de fenêtre privée pour la boîte de dialogue, car vous souhaitez intercepter les messages de la boîte de dialogue avant de passer à la procédure de fenêtre de dialogue par défaut du système.

class peut être, soit :
- un nombre 16 bits
- une chaîne entre guillemets

EXSTYLE *exstyle* — permet de spécifier un style étendu, qui peut être un ou plusieurs des styles étendus suivants ou leur valeur respective séparés par l'opérateur (OR) | (barre verticale) :

WS_EX_DLGMODALFRAME 1h
a une double bordure

WS_EX_NOPARENTNOTIFY 4h
lorsque la fenêtre est créée ou détruite, n'envoie pas le message WM_PARENTNOTIFY à la fenêtre parent

WS_EX_TOPMOST 8h
place la fenêtre au-dessus de toutes celles qui ne sont pas topmost et reste au-dessus d'elles, même si ladite fenêtre est désactivée

WS_EX_ACCEPTFILES 10h
accepte les glisser-déplacer de fichiers

WS_EX_TRANSPARENT 20h
La fenêtre ne doit pas être peinte avant que les frères et sœurs situés sous la fenêtre (créés par le même thread) aient été peints. La fenêtre apparaît transparente car les bits des fenêtres sœurs sous-jacentes ont déjà été peintes.

WS_EX_MDICHILD 40h

une fenêtre enfant MDI

WS_EX_TOOLWINDOW 80h
pour utilisaton en tant que barre d'outils flottante

WS_EX_WINDOWEDGE 100h
bordure apparente

WS_EX_CLIENTEDGE 200h
bordure avec bord creux

WS_EX_OVERLAPPEDWINDOW 300h
Correspond à WS_EX_WINDOWEDGE | WS_EX_CLIENTEDGE

WS_EX_PALETTEWINDOW 188h
Correspond à WS_EX_WINDOWEDGE | WS_EX_TOOLWINDOW | WS_EX_TOPMOST

WS_EX_CONTEXTHELP 400h
la barre de titre inclut un point d'interrogation à utiliser avec le traitement des messages WM_HELP pour afficher l'aide

WS_EX_LEFT 0h
propriété par défaut d'alignement à gauche

WS_EX_RIGHT 1000h
propriété d'alignement à droite pour les langues prescrivant un sens de lecture.

WS_EX_LTRREADING 0h
propriétés par défaut de sens de lecture de gauche à droite

WS_EX_RTLREADING 2000h
propriétés de sens de lecture de droite à gauche pour les langues prescrivant un sens de lecture

WS_EX_RIGHTSCROLLBAR 0h
position par défaut à droite de la zone client

WS_EX_LEFTSCROLLBAR 4000h
position à gauche de la zone client pour les langues prescrivant un sens de lecture

WS_EX_CONTROLPARENT 10000h
les fenêtres enfants sont incluses dans la boîte de dialogue de navigation

WS_EX_STATICEDGE 20000h
la bordure a un style en trois dimensions

WS_EX_APPWINDOW 40000h
fenêtre de niveau supérieur lorsque les affichages sont visibles dans la barre des tâches

WS_EX_LAYERED 80000h
fenêtre en couches généralement avec une forme complexe/animée ou un mélange alpha

WS_EX_NOINHERITLAYOUT 100000h
la disposition de la fenêtre n'est pas transmise aux fenêtres enfants

WS_EX_LAYOUTRTL 400000h
origine horizontale sur le bord droit de la fenêtre pour les langues prenant en charge le sens de lecture

WS_EX_COMPOSITED 2000000h
Peint tous les descendants d'une fenêtre dans l'ordre de peinture du bas vers le haut en utilisant le double tampon.

WS_EX_NOACTIVATE 8000000h
si une fenêtre de premier niveau, ne devient pas la fenêtre de premier plan ou apparaît sur la barre des tâches

FONT *pointsize, "typeface"* ; si DIALOG

FONT *pointsize, "typeface"* [,*poids, italique, jeu caractères*] ; si DIALOGEX

cette instruction doit être présente si la boîte de dialogue a le style DS_SETFONT et contient l'information que le système utilise pour écrire le texte dans la boîte de dialogue et ses contrôles. Si cette instruction est présente, la boîte de dialogue reçoit automatiquement le style DS_SETFONT.r

pointsize nombre 16 bits donnant la taille de la police.

typeface chaîne entre guillemets donnant le nom de la police

poids (DIALOGEX seulement) est soit:
- un nombre 16 bits
- un des mots reconnus par le Compilateur de Ressources et énumérés

ci-dessous (le poids est donné dans la deuxième colonne):

FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_HEAVY	900

si ce paramètre est omis, il est mis à zéro (ne vous en souciez pas)

italique (DIALOGEX uniquement) indique si les italiques doivent être utilisés et est :
- le nombre 0 ou le mot FAUX pour indiquer *non italique*, ou
- le nombre 1 ou le mot TRUE pour indiquer *italique*

si ce paramètre est omis, il est mis à zéro (*non italique*)

charset (DIALOGEX uniquement) indique le jeu de caractères de la boîte de dialogue et est soit :
- un nombre de 8 bits
- une expression dont le résultat est un nombre de 8 bits
- l'un des mots reconnus par le Compilateur de Ressources et énumérés ci-dessous (la valeur est donnée dans la deuxième colonne):

ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
HANGEUL_CHARSET	129
HANGUL_CHARSET	129
GB2312_CHARSET	134
CHINESEBIG5_CHARSET	136
OEM_CHARSET	255
JOHAB_CHARSET	130
HEBREW_CHARSET	177
ARABIC_CHARSET	178
GREEK_CHARSET	161
TURKISH_CHARSET	162
VIETNAMESE_CHARSET	163
THAI_CHARSET	222
EASTEUROPE_CHARSET	238
RUSSIAN_CHARSET	204
MAC_CHARSET	77
BALTIC_CHARSET	186

Si ce paramètre est omis, il est mis à zéro (ne vous en souciez pas)

MENU *menuname* cette instruction, si elle est présente, spécifie un menu pour la boîte de dialogue et *menuname* est, soit :
- un nombre de 16 bits identifiant le menu (ce pourrait être le numéro donné en tant que *nameID* de la ressource du menu)
- une expression dont le résultat est un tel nombre (ou le nom d'une ressource de menu si le *nameID* du menu était un nom)

STYLE *styles* cette instruction est utilisée pour changer le style de boîte de dialogue par défaut qui est :
WS_POPUP | WS_BORDER | WS_SYSMENU et qui correspond à :
80000000h | 800000h | 80000h

Pour supprimer un style par défaut non désiré, vous pouvez utiliser le mot NOT ou le symbole ! (point d'exclamation). Par exemple STYLE NOT SYSMENU ou STYLE! 80000h produit une boîte de dialogue avec, seulement, les styles WS_POPUP et WS_BORDER.

Les styles de dialogue suivants et leurs valeurs correspondantes peuvent être utilisés dans cette instruction :

DS_ABSALIGN	1h

les coordonnées de la boîte de dialogue sont des coordonnées d'écran par opposition aux coordonnées du client

DS_SYSMODAL 2h
donne à la boîte de dialogue le style WS_EX_TOPMOST

DS_FIXEDSYS 8h
utiliser la police fixe du système plutôt que la police proportionnelle

DS_NOFAILCREATE 10h
construire la boîte de dialogue même s'il y a des erreurs dans la fabrication des contrôles

DS_SETFONT 40h
recherche l'instruction FONT pour la police à utiliser dans les contrôles et le texte de la boîte de dialogue

DS_MODALFRAME 80h
crée une boîte de dialogue modale qui peut être combiné avec les styles WS_CAPTION et WS_SYSMENU

DS_NOIDLEMSG 100h
supprime les messages WM_ENTERIDLE que le système pourrait envoyer autrement au propriétaire de la boîte de dialogue

DS_SETFOREGROUND 200h
système pour appeler *SetForegroundWindow* lorsque la boîte de dialogue est créée

DS_CONTROL 400h
fait fonctionner la boîte de dialogue comme la fenêtre enfant d'une autre boîte de dialogue

DS_CENTER 800h
centre la boîte de dialogue aussi loin que possible sur l'écran

DS_CENTERMOUSE 1000h
centre le curseur de la souris dans la boîte de dialogue

DS_CONTEXTHELP 2000h
ajoute un point d'interrogation à la barre de titre pour le traitement de l'aide

Les styles de fenêtres suivants et leurs valeurs respectives peuvent également être utilisés dans cette instruction :

WS_TILED *idem* WS_OVERLAPPED

WS_OVERLAPPED 0h
fenêtre superposée avec une barre de titre et une bordure

WS_MAXIMIZEBOX 10000h
met une coche de maximisation de boîtede dialogue dans la barre de titre

WS_MINIMIZEBOX 20000h
met une coche de minimisation de boîtede dialogue dans la barre de titre

WS_THICKFRAME *idem* WS_SIZEBOX

WS_SIZEBOX 40000h
génère une bordure de cadre épaisse permettant le redimensionnement

WS_SYSMENU 80000h
a un menu système dans sa barre de titre

WS_HSCROLL 100000h
a une barre de défilement horizontale

WS_VSCROLL 200000h
a une barre de défilement verticale

WS_DLGFRAME 400000h
crée un type de cadre de boîte de dialogue modale pour la fenêtre qui ne peut pas avoir une barre de titre

WS_BORDER 800000h
crée une ligne de bordure fine

WS_CAPTION 0C00000h
équivaut à WS_BORDER | WS_DLGFRAME et possède, en plus, une barre de titre

WS_TILEDWINDOW *idem* WS_OVERLAPPEDWINDOW

WS_OVERLAPPEDWINDOW 0C80000h
équivaut à WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | \
WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX, c'est-à-dire une fenêtre superposée combinée à d'autres styles

WS_MAXIMIZE 1000000h
fenêtre créée initialement en format maximisé

WS_CLIPCHILDREN 2000000h
une fenêtre parente peut utiliser ce style pour exclure des zones de fenêtre enfant lors du dessin dans la fenêtre parent

WS_CLIPSIBLINGS 4000000h
Dissocie les fenêtres enfants les unes des autres; C'est-à-dire que lorsqu'une fenêtre enfant particulière reçoit un message WM_PAINT, le style WS_CLIPSIBLINGS dissocie toutes les autres fenêtres enfants se chevauchant en dehors de la région de la fenêtre enfant à mettre à jour.

WS_DISABLED 8000000h
fenêtre créée initialement créé comme désactivée

WS_VISIBLE 10000000h
La fenêtre est initialement visible. Ce style peut être activé et désactivé à l'aide des fonctions *ShowWindow* ou *SetWindowPos*.

WS_MINIMIZE 20000000h
fenêtre créée initialement en format minimisé

WS_CHILD 40000000h
fenêtre enfant qui ne peut pas avoir de barre de menu et ne peut pas être utilisée avec le style WS_POPUP

WS_POPUP 80000000h
fenêtre pop-up qui ne peut pas être utilisée avec le style WS_CHILD

WS_POPUPWINDOW 80880000h
équivaut à WS_POPUP | WS_BORDER | WS_SYSMENU, c'est-à-dire une fenêtre pop-up, utilisée avec WS_CAPTION pour rendre le menu de la fenêtre visible

iconcontrol affiche une icône dans la boîte de dialogue. La syntaxe d'un *iconcontrol* est la suivante :

ICON *nameID, id, x, y*

Où,

nameID est l'identifiant d'une ressource ICON et peut être :
- un nombre de 16 bits de valeur 1 à 7FFFh
- une expression dont le résultat est un nombre 16 bits
- une chaîne entre guillemets contenant le nom de l'icône si celle-ci est identifiée par son nom

id peut être, soit :
- un nombre de 16 bits (DIALOG) ou un nombre 32 bits (DIALOGEX)
- une expression dont le résultat est un nombre tel que décrit ci-dessus

x est la position horizontale de l'icône

y est la position verticale de l'icône

textcontrol £££ est un contrôle de dialogue qui contient du texte et dont le nom est reconnu par le Compilateur de Ressources. Le nom du type de contrôle oblige le compilateur de ressources à attribuer au contrôle la classe correcte et un style par défaut. Pour supprimer un style par défaut, vous devrez utiliser le spécificateur NOT, comme vous le feriez pour supprimer un style par défaut pour la boîte de dialogue elle-même.

La syntaxe d'un *textcontrol* est :

textcontrol "text", id, x, y, width, height, [,*style*[,*extendedstyle*]]
BEGIN
    data-elements
END

Où,

textcontrol est l'un des noms donnés ci-dessous, listés ici avec leur classe et leur style par défaut :

LTEXT classe statique, SS_LEFT | WS_GROUP
crée un contrôle statique avec le texte aligné à gauche

CTEXT classe statique, SS_CENTER | WS_GROUP
crée un contrôle statique avec du texte centré

RTEXT classe statique, SS_RIGHT | WS_GROUP
crée un contrôle statique avec du texte aligné à droite

GROUPBOX	classe de bouton, BS_GROUPBOX crée un rectangle avec label dans la boîte de dialogue dans lequel des autres contrôles peuvent être groupés
CHECKBOX	classe de bouton, BS_CHECKBOX | WS_TABSTOP crée un contrôle de case à cocher (petit rectangle avec du texte à côté)
AUTOCHECKBOX	classe de bouton, BS_AUTOCHECKBOX | WS_TABSTOP crée un contrôle de case à cocher qui passe automatiquement à coché ou non coché lorsque l'utilisateur clique sur le contrôle
STATE3	classe de boutons, BS_3STATE | WS_TABSTOP crée un contrôle de case à cocher qui a 3 états : coché, non coché et désactivé (grisés)
AUTO3STATE	classe de bouton, BS_AUTO3STATE | WS_TABSTOP crée un contrôle de type STATE3 mais qui bascule automatiquement lorsqu'il est cliqué entre les trois états
RADIOBUTTON	classe de bouton, BS_RADIOBUTTON | WS_TABSTOP crée un contrôle de bouton radio
AUTORADIOBUTTON	classe de bouton, BS_AUTORADIOBUTTON | WS_TABSTOP crée un bouton d'option qui bascule automatiquement lorsque l'on clique dessus
PUSHBUTTON	classe de bouton, BS_PUSHBUTTON | WS_TABSTOP crée un contrôle de bouton-poussoir ordinaire contenant du texte
DEFPUSHBUTTON	classe de bouton, BS_DEFPUSHBUTTON | WS_TABSTOP crée un bouton-poussoir ordinaire contenant du texte et étant actif par défaut (par rapport à PUSHBUTTON notamment) si l'utilisateur appuie sur Entrée
PUSHBOX	classe de bouton, BS_USERBUTTON | BS_CHECKBOX | WS_TABSTOP donne un aspect plat au bouton
"text"	texte apparaissant dans le contrôle
id	peut être, soit : - un nombre 16 bits (DIALOG) ou 32 bits (DIALOGEX) - une expression dont le résultat est un des 2 nombres ci-dessus
x	position horizontale du contrôle exprimée en unités de dialogue
y	position verticale du contrôle exprimée en unités de dialogue
width	largeur du contrôle exprimée en unités de dialogue
height	hauteur du contrôle exprimée en unités de dialogue
style	voir **ci-dessous** pour plus de détails
extendedstyle	voir **ci-dessous** pour plus de détails
data-elements	sont un ou plusieurs éléments de données, appelés "données de création". Un pointeur vers ces données est envoyé à la procédure de dialogue dans le paramètre *lParam* du message WM_CREATE lorsque le contrôle est créé. Les données sont faites en utilisant le même format que pour la ressource RCDATA.

non-textcontrol similaire à un *textcontrol*, sauf qu'il ne contient pas de texte dans la première instance (ceci doit être ajouté au moment de l'exécution).

La syntaxe pour un *non-textcontrol* est :

non-textcontrol id, x, y, width, height, [*,style*[*,extendedstyle*]]
BEGIN
    data-elements
END

Où les paramètres *id*, *x*, *y*, *width*, *height*, *style* et *extendstyle* sont les mêmes que pour *text-control*.

non-textcontrol	est l'un des noms donnés ci-dessous, listés ici avec leur classe et leur style par défaut:
EDITTEXT	classe edit, ES_LEFT | WS_BORDER | WS_TABSTOP crée un contrôle d'édition simple

HEDIT	contrôle d'écriture manuscrite (stylo)
IEDIT	contrôle d'encrage d'édition (stylo)
LISTBOX	classe listbox, LBS_NOTIFY | WS_BORDER crée une listbox de base
SCROLLBAR	classe scrollbar, SBS_HORZ crée une barre de défilement de base
COMBOBOX	classe combobox, CBS_SIMPLE crée un contrôle combobox de base

generic-control contrôle dans lequel vous spécifiez explicitement la classe du contrôle. Il est principalement utilisé pour les classes de contrôle qui n'apparaissent pas dans les classes prédéfinies button, edit, static, listbox, scrollbar ou combobox. Cependant, vous pouvez utiliser la syntaxe *generic-control* pour créer ces contrôles.

La syntaxe d'un *generic-control* est la suivante :

CONTROL *"text", id, class, style, x, y, width, height* [,*style,[extendedstyle*]]
BEGIN
    data-elements
END

Où les paramètres *id*, *x*, *y*, *width*, *height*, *style* et *extendstyle* sont les mêmes que pour *text-control*. Notez qu'il n'y a pas de styles par défaut, autres que WS_CHILD et WS_VISIBLE. Donc *style* et *extendedstyle* devraient être choisis pour s'adapter au contrôle particulier créé.

"text" peut être, soit :
- une chaîne entre guillemets contenant le texte du contrôle ou 2 guillemets consécutifs s'il n'y a pas de texte
- dans le cas d'un contrôle de classe STATIC qui a le style SS_BITMAP, "text" peut être un nombre de 16 bits représentant le bitmap concerné
- une expression dont le résultat est un texte ou un nombre tels que définis ci-dessus

Pour les contrôles qui ne prennent normalement pas de texte initial, (voir les contrôles *non-text* ci-dessus), vous devez inclure une chaîne de texte vide en spécifiant "", pour le paramètre *text*.

class peut être, soit :
- un nombre de 16 bits de valeur 80h à 85h (s'il s'agit de l'un des six premiers dans la liste ci-dessous)
- une expression dont le résultat est un nombre tel que défini ci-dessus
- une chaîne entre guillemets contenant le nom de la classe affichée dans la deuxième colonne de la liste ci-dessous
- un mot affiché dans la première colonne de la liste ci-dessous
- une chaîne entre guillemets (pour une classe définie par l'utilisateur)

La classe peut donc être l'une des suivantes:

BUTTON	"Button"	80h
EDIT	"Edit"	81h
STATIC	"Static"	82h
LISTBOX	"Listbox"	83h
SCROLLBAR	"Scrollbar"	84h
COMBOBOX	"ComboBox"	85h
WC_HEADER	"SysHeader32"	
WC_LISTVIEW	"SysListView32"	
WC_TREEVIEW	"SysTreeView32"	
HOTKEY_CLASS	"msctls_hotkey32"	
UPDOWN_CLASS	"msctls_updown32"	
ANIMATE_CLASS	"SysAnimate32"	
WC_COMBOBOXEX	"ComboBoxEx32"	
WC_TABCONTROL	"SysTabControl32"	
MONTHCAL_CLASS	"SysMonthCal32"	
PROGRESS_CLASS	"msctls_progress32"	
REBARCLASSNAME	"ReBarWindow32"	
TOOLTIPS_CLASS	"tooltips_class32"	

TRACKBAR_CLASS	"msctls_trackbar32"
STATUSCLASSNAME	"msctls_statusbar32"
TOOLBARCLASSNAME	"ToolbarWindow32"
DATETIMEPICK_CLASS	"SysDateTimePick32"
WC_IPADDRESS	"SysIPAddress32"
WC_PAGESCROLLER	"SysPager"
WC_NATIVEFONTCTL	"NativeFontCtl"
DRAGLISTMSGSTRING	"commctrl_DragListMsg"

Exemple de ressource DIALOG

```
AboutBox DIALOG 100, 100, 160, 72
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
FONT 8,"MS Sans Serif"
CAPTION "About it"
BEGIN
    CTEXT "Application", -1, 0, 8, 160, 8
    CTEXT "Version 1.2", -1, 0, 16, 160, 8
    DEFPUSHBUTTON "OK", IDOK, 55,52,50,14
    CONTROL 23h,-1,"Static",SS_BITMAP,9,6,125,41
    ICON "MyIcon" -1, 20, 10,0,0
END
```

Exemple de ressource DIALOGEX

```
0x45 DIALOGEX 0x30, 0x20, 0x200, 0x400, 0x98
CAPTION "goodbye"
MENU mymenu
FONT 10,"Times New Roman",FW_NORMAL,1
BEGIN
    CTEXT "hello", 0x20,0x10,0x10,0x10,0x10,,WS_EX_PALETTEWINDOW, 0x333
    COMBOBOX 0x456,30005,4560,300,20
    BEGIN
        0x888888L
        L"hello folks"
        0x99,0x67,0x5555
    END
    CONTROL "ducks",0x460,"ComboBoxEx32",0x100,30005,4560,300,20
    CONTROL "drakes",0x461,0x85,0x100,30005,4560,300,20
    CONTROL "hens",0x462,"Button",0x100,30005,4560,300,20
    CONTROL "and",0x463,"Own class",0x100,30005,4560,300,20
    CONTROL "chickens",0x464,STATUSCLASSNAME,0x100,30005,4560,300,20
    ICON "ANI_ICON",0x465, 6, 9, 18, 20
END
```

2.13 Définitions des instructions

Définir des instructions donne une définition supplémentaire à chaque ressource. Il existe trois types de définition d'instruction :

- instruction VERSION
- instruction CHARACTERISTICS
- instruction LANGUAGE

Chaque instruction de définition est spécifique à la ressource à laquelle elle se réfère, mais si LANGUAGE est spécifiée en dehors d'une ressource spécifique, elle s'appliquera à toutes les ressources par la suite.

2.13.1 VERSION et CHARACTERISTICS

La syntaxe est la même pour les deux :

VERSION *valeur*
CHARACTERISTICS *valeur*

Où,

valeur peut être, soit :
- un nombre 32 bits
- une expression dont le résultat est un nombre 32 bits

La *valeur* est stockée uniquement dans le fichier RES et ni dans le fichier OBJ, ni dans le fichier EXE. Elle ne peut donc être récupérée lors de l'exécution. Cependant, elle pourrait être lue par des outils qui fonctionnent avec le fichier RES, comme certains linkers.

2.13.2 LANGUE

La syntaxe de l'instruction LANGUAGE est la suivante :

LANGUAGE *language, sublanguage*

Cette instruction peut apparaître n'importe où dans le script (en dehors d'une section BEGIN ... END). Dans ce cas, le langage spécifié s'applique à toutes les ressources par la suite. Il peut également apparaître entre l'instruction de ressource elle-même et la section BEGIN ... END comme une instruction de définition. Dans ce cas, la langue spécifiée s'applique uniquement à la ressource dans laquelle elle apparaît.

Les valeurs réelles pour la langue et la sous-langue (*language* et *sublanguage*) apparaissent dans WINNT.H. Les valeurs les plus communes pour le langage sont LANG_ENGLISH (valeur 9h), et pour la sous-langue, SUBLANG_ENGLISH_US (valeur 1h), SUBLANG_ENGLISH_UK (valeur 2h), SUBLANG_ENGLISH_AUS (valeur 3h) et SUBLANG_ENGLISH_CAN (valeur 4h). Si aucune langue n'est spécifiée dans votre script de ressources, le compilateur de ressources utilise la langue du système et l'insère dans le fichier de ressources binaires.

2.13.2.1 Constitution de différentes versions linguistiques d'une ressource

Si votre produit est susceptible d'être utilisé dans plusieurs langues, vous devez fournir les différents messages de menu et de boîte de dialogue, les messages d'avertissement et d'erreur écrits dans ces différentes langues. Les fonctionnalités combinées du compilateur de ressources et de l'API permettent raisonnablement d'y parvenir. La manière exacte d'obtenir ce résultat, dans un contexte de ressources, dépend de la configuration souhaitée :

- différents fichiers EXE ou DLL pour chaque version linguistique, ou
- plusieurs langues disponibles à partir du même EXE

La première peut être réalisée à partir de différents scripts de ressources, ou en utilisant les directives conditionnelles, qui demandent au compilateur de ressources de ne compiler que les parties du script de ressources correspondant à la langue désirée (voir les directives conditionnelles, précédemment).

La seconde est possible car l'API *FindResource* renvoie automatiquement le handle de la ressource qui correspond à la langue par défaut du système à l'ordinateur qui exécute le programme. *FindResource* y parvient en obtenant la langue par défaut du système, puis en appelant *FindResourceEx*. Si vous souhaitez spécifier la langue de la ressource à rechercher plutôt que de vous baser sur la langue par défaut du système, vous pouvez appeler *FindResourceEx* en indiquant directement l'ID de la langue.

La fonction *FindResource* peut être utilisée comme suit :

```
PUSH resource type      ; par type, numéro ou par nom s'il s'agit d'une ressource définie par l'utilisateur
PUSH resource nameID    ; utiliser seulement un identifiant de nom et pas un nombre
PUSH hModule            ; mettre hInst ou zéro pour utiliser le processus courant
CALL FindResource
```

La fonction *FindResourceEx* peut être utilisée comme suit :

```
PUSH resource language  ; voir ci-dessous
PUSH resource nameID    ; utiliser seulement un identifiant de nom et pas un nombre
PUSH resource type      ; par numéro de type ou par nom en cas de ressource définie par l'utilisateur
PUSH hModule            ; mettre hInst ou zéro pour utiliser le processus courant
CALL FindResourceEx
```

Notez que les paramètres *nameID* et de type sont inversés pour *FindResourceEx* !

Le paramètre *resource language* est une valeur de 32 bits composée comme suit :

- 10 bits les moins significatifs : valeur de la langue principale:
- 6 bits suivants : valeur de la langue secondaire
- Les 16 bits les plus significatifs : zéro

Pour connaître la liste des valeurs attribuées à chaque langue, voir WINNT.h

Au retour de *FindResource* et *FindResourceEx*, on obtient le handle nécessaire pour l'appel à *LoadResource*. Vous pouvez avoir plusieurs versions d'un menu ou d'une boîte de dialogue, par exemple avec le même *nameID* mais avec différentes langues spécifiées. *FindResource* retournera le handle de la langue appropriée.

2.13.2.2 Menus dans une langue spécifique

Comme la combinaison de *FindResource* (ou *FindResourceEx*) et *LoadResource* fournit une adresse des données de la ressource elle-même, afin de charger un menu spécifique à la langue, vous devez transmettre cette adresse en tant que modèle de menu à *LoadMenuIndirect*. Vous ne pouvez pas utiliser *LoadMenu* pour charger un menu spécifique à une langue.

2.13.2.3 Dialogues dans une langue spécifique

De même, pour créer une boîte de dialogue dans une langue spécifique, vous devez transmettre l'adresse du modèle de boîte de dialogue retournée par *LoadResource* à *DialogBoxIndirect* (modal) ou *CreateDialogIndirect* (non-modal) au lieu d'utiliser les API *CreateDialog* et *DialogBox*. D'autres composants spécifiques à la langue qui sont nécessaires à votre boîte de dialogue, soit reçoivent un *nameID* unique, soit sont chargés au moment de l'exécution. Par exemple, si vous avez une version française de votre boîte de dialogue et que vous souhaitez inclure un menu français au moyen de MENU *defining-statement* dans le script de ressource, un *nameID* doit être donné dans l'instruction MENU qui fait uniquement référence au menu français. Sinon, si vous avez plusieurs versions linguistiques du menu toutes avec le même *nameID*, vous pouvez charger le menu à l'exécution sur le message INIT_DIALOG. Cela se fait en appelant les fonctions *FindResource* ou *FindResourceEx*, puis *LoadResource* (pour obtenir et adresser le modèle de menu approprié du menu), *LoadMenuIndirect* (pour obtenir le handle de menu) et *SetMenu* (pour charger le menu dans la boîte de dialogue).

2.13.2.4 Chaînes et données brutes spécifiques à la langue

Lors du chargement de chaînes spécifiques à la langue, vous pouvez utiliser directement *LoadString*. Cela charge une chaîne en fonction de la langue par défaut du système. Alternativement, l'adresse de la chaîne elle-même peut être obtenue en utilisant *FindResource*, ou *FindResourceEx* et *LoadResource*. La même méthode fonctionne pour RCDATA et pour une ressource définie par l'utilisateur (et, en effet, pour toute ressource qui peut être utilisée en obtenant une adresse des données brutes de la ressource).

2.13.2.5 Exemples de visualisation

Testbug.Exe fournit des exemples de chargement de menus, de boîtes de dialogue, de chaînes et de données brutes spécifiques à une langue. Les fonctions pertinentes à examiner avec le débogueur sont :

- **Menus, boîtes de dialogue et exemple de données brutes** : cliquez sur "use of resources/different language versions/various resources in English" oo "same showing French versions" dans le menu, en regardant avec le débogueur les fonctions RESOURCES_IN_ENGLISH et RESOURCES_IN_FRENCH qui créent des boîtes de dialogue utilisant le. dlg wndproc appelé MDLangDlgProc.
- **Chaînes** : cliquez sur "Language specific string" dans le menu, en scrutant, avec le débogueur, la fonction STRINGTABLES_BY_LANGUAGE.

2.13.2.6 Critères de recherche de langue

Lors de la recherche de ressources spécifiques à une langue, le système recherche une correspondance de langue, mais admet certains écarts si aucune correspondance exacte n'est trouvée. Le système effectue la recherche d'une ressource de type et d'ID corrects dans l'ordre ci-dessous, en descendant la liste tant que la ressource n'est pas encore trouvée :

- Correspondance simultanée de la langue primaire et de la sous-langue
- correspondance dans la langue primaire seulement
- Une langue primaire de LANG_NEUTRAL (valeur zéro)
- Une langue primaire de l'anglais
- N'importe quelle langue

De toute évidence, si la langue spécifiée exacte n'est pas présente dans l'EXE, *FindResourceEx* fournira une sortie à condition que le type de ressource et l'ID correspondent. Il n'y a donc pas lieu de s'inquiéter du fait que votre produit ne fonctionne pas si vous utilisez des ressources spécifiques à la langue.

2.13.2.7 Valeurs de langue personnalisées

Vous pouvez spécifier vos propres valeurs de langue si vous le souhaitez, et par cette méthode il sera possible de faire plusieurs versions de menus, dialogues et chaînes pour votre produit dans la *même* langue si vous le souhaitez, en utilisant la sélection de langue automatique ci-dessus. Pour que le système reste opérationnel, assurez-vous que la valeur de votre *langue* soit comprise entre 200h et 3FFh et que la valeur de la *sous-langue* soit comprise entre 20h et 3Fh. Ces valeurs sont soigneusement choisies. Le véritable languageID inséré dans le fichier de ressources binaires et conservé dans le fichier EXE n'a que 16 bits, la valeur du langage occupant les 10 bits

les plus significatifs (d'où une valeur maximale de 3FFh) et la valeur de *sous-langue* occupant les 6 bits les moins significatifs d'où une valeur maximale de 3Fh).

Exemples d'instruction LANGUAGE

```
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_UK
456h RCDATA
BEGIN
    "Hello world (in English)\0"
END

456h RCDATA
LANGUAGE LANG_FRENCH, SUBLANG_FRENCH
BEGIN
    "Bonjour le monde (en Français)\0"
END

STRINGTABLE
BEGIN
    55h "An English string"
END

LangMenu MENU
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_UK
BEGIN
    MENUITEM "English menu item" 22h
END

LangMenu MENU
LANGUAGE LANG_FRENCH, SUBLANG_FRENCH
BEGIN
    MENUITEM "Version française" 22h
END
```

2.14 Mise en majuscules des nameIDs et types

Nous avons vu qu'un *nameID* pouvait être une chaîne de caractères ou un nombre. Pour chaque type de ressource avec une langue particulière, *nameID* doit être unique. C'est-à-dire que deux boîtes de dialogue peuvent avoir le même *nameID* à condition qu'elles aient des valeurs de langue différentes. Nous avons également vu qu'une ressource définie par l'utilisateur aura un type de ressource identifié par une chaîne de caractères ou un nombre.

Cela dit, le Compilateur de Ressources convertira toujours le *nameID* ou le *type* en majuscules en les insérant dans le fichier RES ou le fichier OBJ. La raison en est que cela est attendu par Windows. En pratique, cela n'affecte pas la façon dont vous utilisez la ressource, car le système va également mettre en majuscules tous les *nameID* ou *types* à destination des API gérant des ressources. Par exemple supposons que vous ayez une boîte de dialogue avec le *nameID* "MyDlg". Pour utiliser la boîte de dialogue, vous allez passer la chaîne "MyDlg" à l'API *CreateDialog*. Mais en fait, le système convertira la chaîne en MYDLG et recherchera ensuite cette chaîne, ainsi convertie, dans l'exécutable.

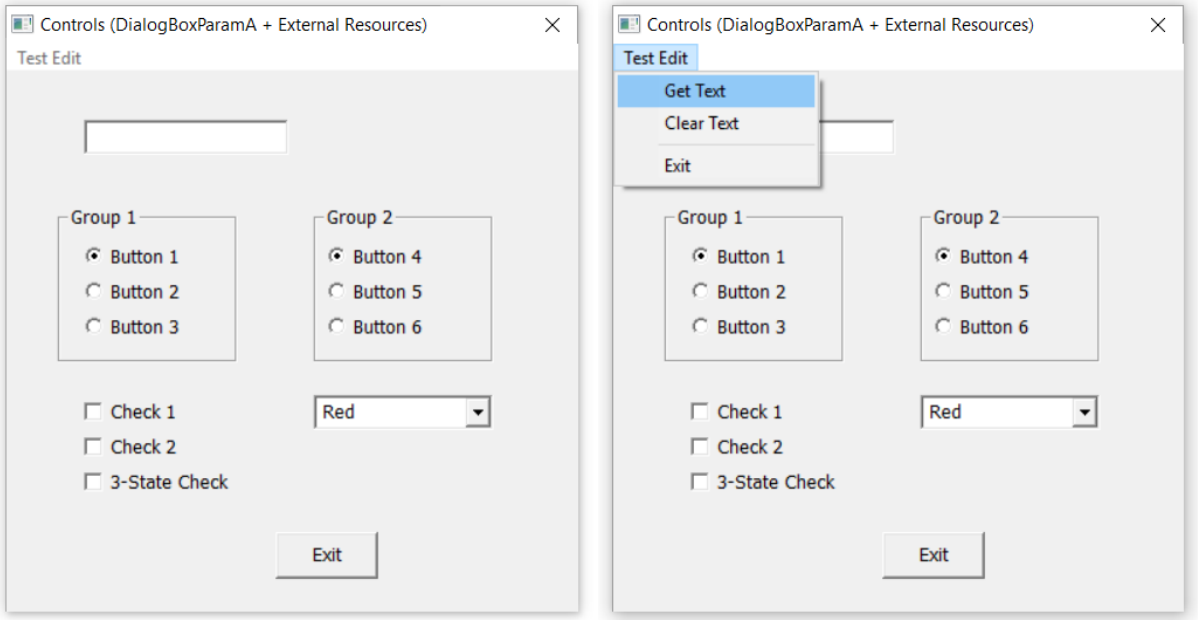
Cela a deux implications que vous ne devez pas ignorer :

- Premièrement, cela signifie qu'il y aurait un conflit dans l'exemple ci-dessus si vous aviez, d'aventure, une deuxième boîte de dialogue (avec la même valeur de langue) appelée MYDLG. Cela serait identifié au moment de la phase d'édition de liens.
- Deuxièmement, le Compilateur de Ressources doit s'assurer que non seulement les *nameID* anglais, mais également les types définis par l'utilisateur sont convertis correctement en majuscules. Ainsi, le compilateur de ressources s'appuie-t-il sur l'API *CharUpper* pour effectuer cette conversion. Cette API peut gérer un certain nombre de langues correctement, mais elle repose sur le mappage linguistique. Assurez-vous que la machine utilisée pour la compilation prend en charge la langue utilisée dans les *nameID* et les types définis par l'utilisateur. Beaucoup de langues n'ont pas de majuscules, auquel cas aucune conversion n'aura lieu.

2.15 Exemple complet de programme avec ressources

Il peut être intéressant de décrire la conception complète d'un programme mettant en œuvre des ressources pour créer une boite de dialogue, des contrôles et un menu à l'aide d'un fichier de ressources. On trouvera également un certain intérêt à rapprocher ce programme de celui de l'annexe A du volume 1 de cette série. Décrit sous le

nom *HelloDialog.asm*, ce dernier utilise en effet une autre variante de la fonction DialogBox – *DialogBoxIndirectParamA* – permettant d'intégrer le modèle dans le fichier source au lieu de le loger dans un fichier de ressources séparé.



Dans la figure ci-dessus, la capture d'écran de gauche montre l'affichage tel qu'il apparaît lors du lancement du programme. Celle de droite montre le menu en action. Les contrôles suivants sont implémentés :

- Contrôle d'Edition mono-ligne
- Controles Auto-Radio Button répartis en 2 groupes,
- Contrôles de CheckBox dont une à 3 états
- Contrôle Combo Box en version Drop-Down.

Le programme se compose de 2 scripts : le premier, décrivant le corps du programme à compiler sous GoAsm ; le second incluant le fichier de ressources devant être compilé sous GoRC. Tout naturellement, l'édition des liens des fichiers *.obj* et *.res* résultants est assurée par GoLink.

2.15.1 Le script de ressources

2.15.1.1 Listing

```
//------------------------------
//      DialogBoxRes.rc
//------------------------------

#include "resource.h"

#define ID_EDIT         900
#define ID_GRB1         910
#define ID_AUTRD1       911
#define ID_AUTRD2       912
#define ID_AUTRD3       913
#define ID_GRB2         920
#define ID_AUTRD4       921
#define ID_AUTRD5       922
#define ID_AUTRD6       923
#define ID_AUTCHK1      931
#define ID_AUTCHK2      932
#define ID_AUTCHK3      933
#define ID_EXIT         950
#define ID_COMBO        940
#define MY_DLG          4000
#define MY_MNU          4001

#define IDM_GETTEXT     32000
```

```
#define IDM_CLEAR       32001
#define IDM_EXIT        32002

MY_DLG DIALOG 50, 30, 250, 200
STYLE DS_CENTER|WS_SYSMENU
FONT 10, "Tahoma"
CAPTION "Some Controls by DialogBoxParamA and Resources"
MENU MY_MNU

BEGIN
        EDITTEXT ID_EDIT, 30,16,80,12
        GROUPBOX "Group 1", ID_GRB1, 20, 45, 70, 54
        AUTORADIOBUTTON "Button 1", ID_AUTRD1, 30, 55, 50, 16, WS_GROUP
        AUTORADIOBUTTON "Button 2", ID_AUTRD2, 30, 67, 50, 16
        AUTORADIOBUTTON "Button 3", ID_AUTRD3, 30, 79, 50, 16

        CONTROL "Group 2", ID_GRB2, "Button", BS_GROUPBOX, 120, 45, 70, 54
        CONTROL "Button 4", ID_AUTRD4, "Button", BS_AUTORADIOBUTTON|WS_GROUP, 125, 55, 50, 16
        CONTROL "Button 5", ID_AUTRD5, "Button", BS_AUTORADIOBUTTON, 125, 67, 50, 16
        CONTROL "Button 6", ID_AUTRD6, "Button", BS_AUTORADIOBUTTON, 125, 79, 50, 16

        AUTOCHECKBOX "Check 1", ID_AUTCHK1, 30, 110, 50, 16
        AUTOCHECKBOX "Check 2", ID_AUTCHK2, 30, 122, 50, 16
        AUTO3STATE "3-State Check", ID_AUTCHK3, 30, 134, 60, 16

        COMBOBOX ID_COMBO,120,112,70,40,CBS_DROPDOWN|CBS_HASSTRINGS|WS_TABSTOP

        PUSHBUTTON "Exit", ID_EXIT, 105, 170, 40, 16
END

MY_MNU  MENU
BEGIN
        POPUP "Test Edit"
        BEGIN
                MENUITEM "Get Text", IDM_GETTEXT
                MENUITEM "Clear Text", IDM_CLEAR
                MENUITEM  SEPARATOR
                MENUITEM "E&xit", IDM_EXIT
        END
END
```

2.15.1.2 Analyse du script de ressources

La fonction *DialogBoxParamA* ne permet qu'un accès à des ressources externes. Ce sera donc l'accès à une ressource DIALOG qui permet de paramétrer l'instruction *DialogBoxParamA* du programme principal et d'y raccrocher un MENU avec la directive MENU MY_MNU (Il sera question plus loin du du menu en tant que tel labelisé MY_MNU).

```
MY_DLG DIALOG 50, 30, 250, 200
…
MENU MY_MNU
```

Tous les contrôles sont définis dans le process BEGIN / END qui suit. Les différents contrôles se présentent comme suit :

- Contrôle d'Edition : `EDITTEXT ID_EDIT, 30,16,80,12`
  Il est constitué pour ne supporter qu'une seule ligne de texte. Ce contrôle reçoit directement les frappes au clavier qui ne transitent donc pas par la procédure de dialogue. Le seul moyen d'interception des saisies au clavier passe par l'emploi de la technique de sous-classement. Du texte peut être envoyé contrôle d'édition avec la fonction *SetDlgItemText*. A l'inverse, le contenu du même contrôle peut être lu par *GetDlgItemText*.
- Contrôle de groupe : `GROUPBOX "Group 1", ID_GRB1, 20, 45, 70, 54`
  Il a pour seule fonction de dessiner un rectangle dans la zone client constitué d'une ligne de faible épaisseur. Le texte du contrôle ("Group 1") est inséré sur la partie gauche de la ligne supérieure de ce rec-

tangle. Notez que ce contrôle est purement graphique et n'a aucune action sur les contrôles Autoradio qu'il englobe, notamment en ce qui concerne leur appartenance à tel ou tel groupe.

- Premier groupe de boutons Autoradio :

```
AUTORADIOBUTTON "Button 1", ID_AUTRD1, 30, 55, 50, 16, WS_GROUP
AUTORADIOBUTTON "Button 2", ID_AUTRD2, 30, 67, 50, 16
AUTORADIOBUTTON "Button 3", ID_AUTRD3, 30, 79, 50, 16
```

  Nous utilisons ici la ressource AUTORADIOBUTTON pour définir un contrôle Autoradio. C'est le style WS_GROUP qui détermine le premier bouton du Groupe 1. La fin du groupe sera déterminée par la déclaration du style WS_GROUP sur le premier bouton du groupe suivant ou par le simple fait de déclarer un bouton d'un autre type tel que AUTOCHECKBOX par exemple.

- Deuxième groupe de boutons Autoradio :

```
CONTROL "Button 4", ID_AUTRD4, "Button", BS_AUTORADIOBUTTON|WS_GROUP, 125, 55, 50, 16
CONTROL "Button 5", ID_AUTRD5, "Button", BS_AUTORADIOBUTTON, 125, 67, 50, 16
CONTROL "Button 6", ID_AUTRD6, "Button", BS_AUTORADIOBUTTON, 125, 79, 50, 16
```

  Nous aurions pu écrire ces contrôles Autoradio comme précédemment mais nous avons voulu illustrer ici une utilisation de la forme générique de ressource permettant de définir la plupart des contrôles :

  CONTROL *"text", id, class, style, x, y, width, height* [,*style,[extendedstyle*]]

  Autant dire que c'est plus complexe et donc, sans intérêt avec, de surcroît, une lisibilité moindre. Mais cela méritait d'être mentionné.

- Contrôles CheckBox (case à cocher) :

```
AUTOCHECKBOX "Check 1", ID_AUTCHK1, 30, 110, 50, 16
AUTOCHECKBOX "Check 2", ID_AUTCHK2, 30, 122, 50, 16
AUTO3STATE "3-State Check", ID_AUTCHK3, 30, 134, 60, 16
```

  Nous utilisons ici la ressource AUTOCHECKBOX pour définir les 2 premiers contrôles CheckBox qui consistent en des cases à cocher. La ressource AUTO3STATE est une case à cocher 3 états, comme son nom l'indique. Le troisième état est une coche grisée.

- Contrôles Combo Box : `COMBOBOX ID_COMBO,120,112,70,40,CBS_DROPDOWN|CBS_HASSTRINGS|WS_TABSTOP`
  Une combo box se compose d'une liste et d'un champ de sélection. La liste présente les options qu'un utilisateur peut sélectionner et le champ de sélection affiche la sélection en cours. Nous adoptons ici la forme Drop-Down justifiant la combinaison de styles CBS_DROPDOWN|CBS_HASSTRINGS. Le style WS_TABSTOP a été ajouté ici pour faire en sorte que le déplacement du focus d'entrée sur les différents contrôles au moyen de la touche TAB n'évite pas la Combo Box.

Le menu est défini par les instructions qui suivent :

```
MY_MNU  MENU
BEGIN
     POPUP "Test Edit"
     BEGIN
          MENUITEM "Get Text", IDM_GETTEXT
          MENUITEM "Clear Text", IDM_CLEAR
          MENUITEM  SEPARATOR
          MENUITEM "E&xit", IDM_EXIT
     END
END
```

L'instruction MENUITEM SEPARATOR trace une ligne de séparation entre les items "Clear Text" et "Exit".

2.15.2 Le script du programme principal

2.15.2.1 Listing

```
;-------------------------
;    DialogBoxRes.asm
;-------------------------
;=== Windows Equates ===
WM_CREATE     = 1h
WM_INITDIALOG = 110h
WM_DESTROY    = 2h
WM_COMMAND    = 111h
WM_PAINT      = 0Fh
BN_CLICKED    = 0h
CS_VREDRAW    = 1h
```

```
CS_HREDRAW     = 2h
CS_CLASSDC     = 40h
NULL           = 00h
TRUE           = 1h
FALSE          = 0h

WS_POPUP       = 80000000h
WS_VISIBLE     = 10000000h
WS_CAPTION     = 0C00000h
WS_SIZEBOX     = 40000h
WS_SYSMENU     = 80000h
WS_MINIMIZEBOX= 20000h
WS_MAXIMIZEBOX= 10000h
SW_SHOWDEFAULT= 10h
MB_OK          = 0h
CB_ADDSTRING   = 143h
CB_SETCURSEL   = 14Eh
CBN_CLOSEUP    = 8h

;=== User Equates ===
ID_EDIT        =  900
ID_GRB1        =  910
ID_AUTRD1      =  911
ID_AUTRD2      =  912
ID_AUTRD3      =  913
ID_GRB2        =  920
ID_AUTRD4      =  921
ID_AUTRD5      =  922
ID_AUTRD6      =  923
ID_AUTCHK1     =  931
ID_AUTCHK2     =  932
ID_AUTCHK3     =  933
ID_EXIT        =  950
ID_COMBO       =  940
IDC_BUTTON     = 3001
IDC_EXIT       = 3002

MY_DLG         = 4000

IDM_GETTEXT    = 32000
IDM_CLEAR      = 32001
IDM_EXIT       = 32002
;=====================================================================
DATA section
;---------------------------------------------------------------------
hInstance    DD 0
hDlg         DD 0
hEdt         DD 0
hCombo       DD 0
Item1        DB 'Red',0
Item2        DB 'Green',0
Item3        DB 'Yellow',0
DlgName      DB "MyDialog", 0
AppName      DB "Text", 0
buffer       DB 512 Dup ?

;=====================================================================
CODE section
;---------------------------------------------------------------------
start:
     Invoke GetModuleHandleA, NULL
     Mov [hDlg], Eax
```

```
     Invoke DialogBoxParamA, [hDlg], MY_DLG, NULL, Addr DlgProc,NULL
     Invoke ExitProcess, Eax

;-------------------------------------------------------------------------
; PROCÉDURE DE DIALOGUE
;-------------------------------------------------------------------------
DlgProc Frame hDlg, uMsg, wParam, lParam
     Uses Ebx, Edi, Esi

     Mov Eax, [uMsg]
     Cmp Eax, WM_INITDIALOG
          Jne > Cmde
          Invoke CheckDlgButton,[hDlg],ID_AUTRD1,1     ; activation AutoRadioButton n°1 (Groupe 1)
          Invoke CheckDlgButton,[hDlg],ID_AUTRD4,1     ; activation AutoRadioButton n°4 (Groupe 2)

          Invoke GetDlgItem,[hDlg],ID_EDIT             ; récupération du handle du contrôle Edit
          Mov [hEdt],Eax                               ;   à partir de l'ID déclaré dans la ressource
          Invoke GetDlgItem,[hDlg],ID_COMBO            ; récupération du handle du contrôle combo box
          Mov [hCombo],Eax                             ;   à partir de l'ID déclaré dans la ressource

;------------ Chargement des items dans la combo Box -----------
          Invoke SendMessageA,[hCombo],CB_ADDSTRING,0,offset Item1   ; envoi Item1 à la Combo Box
          Invoke SendMessageA,[hCombo],CB_ADDSTRING,0,offset Item2   ; envoi Item2 à la Combo Box
          Invoke SendMessageA,[hCombo],CB_ADDSTRING,0,offset Item3   ; envoi Item3 à la Combo Box
          Invoke SendMessageA,[hCombo],CB_SETCURSEL,0,0              ; positionnement de l'Item 1 (0)
                                                                     ; dans la zone d' édition
          Mov Eax, TRUE
          Jmp >>.return
     Cmde:
     Cmp Eax, WM_COMMAND
          Jne >>.false
          Mov Eax, [wParam]
          Cmp D[lParam], 0
          Jne > CdeCtrl
;------------ Commandes items du Menu -----------
               Cmp Eax, IDM_GETTEXT                    ; activation item "Get Text" du menu
               Jne >
               Invoke GetDlgItemTextA, [hDlg], ID_EDIT, Addr buffer, 512   ; contenu Edit dans buffer
               Invoke MessageBoxA, NULL, Addr buffer, Addr AppName, MB_OK  ; affichage boîte message
                                                       ; avec contenu buffer
               jmp >>.true
:
               Cmp Eax, IDM_CLEAR                      ; activation item "Clear Text" du menu
               Jne >
               Invoke SetDlgItemTextA, [hDlg], ID_EDIT, NULL
               jmp >.true
:              ; on considère ici qu'il s'agit par défaut de l'item "Exit" du menu
               ; (message IDM_EXIT)
                    invoke EndDialog,[hDlg],NULL
                    jmp >.true

;------------ Commandes des contrôles -----------
          CdeCtrl:
          Mov Edx, [wParam]
          Shr Edx,16                                   ; fait en sorte que DX récupère les 16 bits
                                                       ; de poids fort de wParam
          Cmp Dx, BN_CLICKED                           ; clic sur un contrôle bouton ?
          Jne > ComboB
               Cmp Ax, ID_EXIT                         ; bouton Exit
               Jne >
               invoke EndDialog,[hDlg],NULL
```

```
                Jmp >.true
:
                Cmp Ax, AUTRD1                          ; bouton Button 1 (Auto-Radio)
                Jne >
                invoke IsDlgButtonChecked,[hDlg],AUTRD1  ; EAX = état du bouton
                ; <process du bouton>
               Jmp >.true
:
                Cmp Ax, AUTRD2                          ; bouton Button 2 (Auto-Radio)
                Jne > Other_Buttons
                invoke IsDlgButtonChecked,[hDlg],AUTRD2  ; EAX = état du bouton
                ; <process du bouton>
                Jmp >.true

                ;.... <Autres boutons>

           ComboB:
           Cmp Dx, CBN_CLOSEUP                          ; message de fermeture de la ComboBox
           Jne > .true

           Other_Buttons:
           Invoke SetFocus, [hEdt]                  ; re-positionnement du focus d'entrée sur le
                                                    ; contrôle d'édition à la suite d'une action sur
                                                    ; une CheckBox, un RadioButton ou une Combo box

      .true
      Mov Eax,TRUE
      Jmp >.return
      .false
      Xor Eax, Eax
      .return
      Ret
EndF
```

2.15.2.2 Analyse du script du programme principal

Comparé à l'utilisation des API *CreateWindow* et *CreateDialog*, le corps du programme avec la fonction *DialogBoxParam* est extrêmement simple. Il n'y a plus à paramétrer la structure WNDSTRUCT, à l'enregistrer, ni à mettre en œuvre une boucle de message.

```
start:
     Invoke GetModuleHandleA, NULL
     Mov [hDlg], Eax

     Invoke DialogBoxParamA, [hDlg], MY_DLG, NULL, Addr DlgProc,NULL
     Invoke ExitProcess, Eax
```

Certes, le Callback dédié au traitement des messages demeure mais sous une forme sensiblement différente en dépit des apparences. La procédure de fenêtre est assurée par Windows lui-même ce qui fait que le Callback présent dans le programme n'en est que le prolongement. Pour cette raison, on lui donne le nom de Procédure de Dialogue (`DlgProc` ici). Voici sa forme générale dans la syntaxe GoAsm :

```
DlgProc Frame hDlg, uMsg, wParam, lParam
     Uses Ebx, Edi, Esi

     Mov Eax, [uMsg]
     Cmp Eax,WM_INITDIALOG
          Jne > Cmde
;         ...
     Cmde:
     Cmp Eax,WM_COMMAND
          Jne >>.false
;         ...
     .false
     Xor Eax, Eax
```

```
        .return
        Ret
    EndF
```

En raison de l'utilisation de la fonction *DialogBoxParam*, il n'y a plus d'appel à la fonction *DefWindowProc* et la fin du dialogue est résolue par la fonction *EndDialog*. Le fait que la procédure de dialogue ne soit qu'un prolongement de la procédure de fenêtre de Windows induit la nécessité de renvoyer dans EAX une valeur booléenne indiquant que le message a été traité (TRUE) ou non (FALSE). Par ce procédé, tout message non traité dans la procédure de dialogue pourra donc l'être par Windows, à la manière de *DefWindowProc*.

Les actions de la souris – et accessoirement du clavier – sur les contrôles et le menu sont analysées par examen du contenu des paramètres *wParam* et *lParam* du message WM_COMMAND dans la procédure de dialogue.

Les commandes du menu

Sous WM_COMMAND, le programme identifie une commande de menu lorsque *lParam* = 0. Dans ce cas, les 16 bits de poids faible de *wParam* sont réputés contenir l'ID du contrôle ayant subi une action. Ici, IDM_GETTEXT indique que nous avons sétectionné l'item de menu "Get Text". De fait, nous copions le contenu du contrôle d'édition dans un buffer avec la fonction *GetDlgItemTextA* puis affichons ce contenu avec une classique *MessageBoxA*. IDM_CLEAR correspond à l'item de menu "Clear Text" et vide le contenu de la saisie effectuée dans le contrôle d'édition. IDM_EXIT, ID de l'item "Exit", est considéré présent par défaut en l'absence de IDM_GETTEXT et IDM_CLEAR. Il actionne la fonction *EndDialog* qui entraine la sortie de la boîte de dialogue.

Les commandes de contrôles

Toujours sous WM_COMMAND, le programme identifie une commande de contrôle lorsque *lParam* est différent de zéro. Dans ce cas, les 16 bits de plus fort poids de *wParam* contiennent le code de notification du contrôle ayant subi une action. Ce code rend compte de la manière dont est sollicité le contrôle. Un simple clic sur un contrôle est identifié par la valeur BN_CLICKED. Un double clic l'est par la valeur BN_DBLCLK (ou BN_DOUBLECLICKED). Quant au code de notification CBN_CLOSEUP (que nous utilisons dans ce programme), il caractérise la fermeture de la Combo Box. Enfin, les 16 bits de plus faible poids de *wParam* contiennent l'ID du contrôle. Les boutons radio, les check box et la combo box ne réalisent aucun traitement sinon de ramener systématiquement le focus d'entrée dans le contrôle d'édition avec la fonction *SetFocus* après chaque action sur l'un de ces contrôles. Seul, le contrôle d'édition a une véritable gestion. Voyons maintenant ce qui se passe sur chacun des contrôles :

- **Contrôles d'édition** (identifiant ID_EDIT) : voir les remarques faites en marge du script de ressources.
- **Contrôles bouton Auto-radio** (identifiants AUTRD1 à AUTRD6) : il s'agit de la forme logique la plus récente des boutons radio. Lorsque les boutons radio sont organisés en groupes (ici, 2 groupes de 3 boutons chacun), un clic sur un bouton n'agit que si ce ce bouton est inactif, auquel cas le petit cercle voit apparaître en son milieu. Simultanément, le bouton qui était actif avant cette action devient inactif à son tour. Mais, contrairement aux apparences, le changement d'état des boutons radio n'a pas de traduction logicielle dans le programme et il faut lire l'état du bouton radio par la fonction :

  ```
  invoke IsDlgButtonChecked,[hDlg],AUTRD1
  ```

  AUTRD1 est l'ID du bouton et l'état de ce dernier est répercuté dans le registre EAX (1 = actif, 0 = inactif).

- **Contrôles Check Box 2 états** (identifiants AUTCHK1 et AUTCHK2) : il s'agit de la forme logique la plus récente des Check Box (cases à cocher). Celles-ci ne sont pas organisées en groupe. Un clic sur une check box fait basculer son état. L'activation se matérialise par l'apparition d'une coche dans le petit carré. Comme pour les boutons radio le changement d'état d'une check box n'a pas de traduction logicielle dans le programme et il faut lire l'état par la fonction :

  ```
  invoke IsDlgButtonChecked,[hDlg],AUTCHK1
  ```

  AUTCHK1 est l'ID de la check box et l'état de cette dernière est répercuté dans le registre EAX (1 = actif, 0 = inactif).

- **Contrôle Check Box 3 états** (identifiant AUTCHK3) : il s'agit d'une Check Box disposant de 3 états. Le troisième est matérialisé par une coche grisée indiquant que l'option est cochée mais qu'elle est ignorée par le programme. Comme pour les 2 autres check box, le changement d'état n'a pas de traduction logicielle dans le programme et il faut lire l'état par la fonction :

  ```
  invoke IsDlgButtonChecked,[hDlg],AUTCHK3
  ```

  AUTCHK3 est l'ID de la check box et l'état de cette dernière est répercuté dans EAX (0 = inactif, 1 = actif, 2 = neutre).

2.15.3 Le fichier .BAT de compilation et d'édition de liens

```
set appname= DialogBoxRes
set dirdll=C:\Windows\System32

GoRC /r %appname%.rc
GoAsm %appname%.asm
```

```
GoLink %appname%.obj %appname%.res %dirdll%\kernel32.dll %dirdll%\user32.dll %dirdll%\gdi32.dll
%dirdll%\Comctl32.dll
del %appname%.obj
pause
```

Ce fichier compile le fichier de ressources, assemble le script du programme puis procède à l'édition des liens.

2.16 Aspects légaux

2.16.1 Copyright

2.16.2 GoRC - licence et distribution

Vous pouvez utiliser GoRC à toutes fins, y compris la réalisation de programmes commerciaux. Vous pouvez le redistribuer librement (mais pas contre paiement ni dans le cadre d'une utilisation avec tout programme ou matériel pour lequel l'utilisateur est invité à payer). Vous n'avez pas le droit de dissimuler ou de refuser mes droits d'auteur.

2.16.3 Avertissement

Je me suis efforcé de faire en sorte que les traitements accomplis par GoRC soient exempts d'erreur. Pour autant, vous l'utilisez entièrement à vos risques et périls et je ne peux accepter aucune responsabilité découlant d'un fonctionnement incorret, d'un traitement erroné ou d'erreurs affectant éventuellement ce manuel.

Chapitre 3 – Débogueur GoBug

3.1 GoBug: Quoi, quand et comment?

3.1.1 Que peut-on faire avec GoBug?

Ce qui suit décrit certaines des choses les plus utiles que vous pouvez faire avec GoBug dans un contexte Windows

GoBug est un débogueur symbolique capable d'exécuter un autre programme (le "debuggee") dans des conditions sous contrôle.

GoBug vous permet les actions suivantes :

- Utilisez **F5** ou **F6** pour parcourir le code du debuggee en pas-à-pas, c'est-à-dire que le processeur exécute une seule instruction à la fois, que vous pouvez observer dans le volet de code. Pendant ce temps, vous pouvez voir comment cela affecte les registres principaux, les flags, les zones de pile et de mémoire du débogueur. Vous pouvez également afficher le volet des registres à virgule flottante, celui des registres MMX ou les volets des registres 3DNow!, XMM, SSE et SSE2.

  ➲ *Essayez le test de saut conditionnel et d'utilisation des flags de Testbug, pour pratiquer le pas à pas unique* (*voir le chapitre 1 du volume 3 sur ce point*)

- Utilisez plus particulièrement **F6** pour suspendre le pas-à-pas pendant l'exécution de la routine d'un CALL dont le détail ne vous intéresse pas.
- Utilisez **F7** pour exécuter le debuggee avec les interceptions d'événements/messages et complètement piégé en vue d'obtenir un journal détaillé de l'exécution du debuggee pour une vue ultérieure. Ce sera un journal très détaillé montrant chaque instruction et montrant les modifications apportées aux registres.
- Utilisez **F8** pour exécuter le debuggee avec les interceptions d'événements/messages mais avec libération du piègeage en vue d'obtenir un journal partiel des actions du debuggee. Ce journal est limité aux événements et aux messages.
- Utilisez **F9** pour exécuter le débuggee en arrière-plan jusqu'à ce qu'une exception se produise (après quoi vous pouvez voir le point de l'exception dans le volet de code et visualiser les registres ainsi que l'historique de la pile).

  ➲ *Pratiquez cet exercice en utilisant Testbug comme debuggee*

- Exécutez le debuggee en arrière-plan (touche F9), puis essayez de reprendre le contrôle en utilisant le raccourci clavier ou le contrôle de signalisation de type routier.

  ➲ *Pratiquez cet exercice en mettant Testbug dans une boucle infinie, puis sortez de la boucle*

- Exécutez mais sautez sur des parties de code en définissant un point d'arrêt de code ou en exécutant le debuggee à la fin de la procédure en cours.
- Exécutez le code du débogueur à un point d'arrêt dans le code, puis arrêtez l'exécution. Cela vous permet d'effectuer des tests à ce moment-là ou d'effectuer une seule étape à partir de ce point pour tester cette partie particulière de votre code.
- Définissez un point d'arrêt pour exécuter le débogueur jusqu'à n'importe quel message ou jusqu'à ce qu'un message particulier, ou jusqu'à un message particulier à une procédure de fenêtre particulière.
- Visualisez la séquence et le détail des messages en explorant en pas-à-pas une procédure ou une API qui déclenche l'activité du message, puis examinez chaque message en utilisant le break de message en pas-à-pas ou en lançant l'exécution (hook, partie log) puis en affichant les messages dans le volet du journal d'événements / messages.

  ➲ *Pratiquez cet exercice en utilisant Testbug comme debuggee*

- Définissez un point d'arrêt pour exécuter le debuggee jusqu'au début d'un nouveau thread, puis activez le mode pas-à-pas pour étudier l'exécution de ce thread.

  ➲ *Pratiquez cet exercice en utilisant Testbug comme debuggee*

- Appliquez le mode pas-à-pas, sur plus d'un thread à la fois pour voir comment Windows partage le temps processeur entre les threads ou, mieux, pour comprendre une application multi-thread. Voir dans le journal l'interaction en temps réel entre les threads.

Pratiquez cet exercice en utilisant Testbug comme debuggee

- Visualisez en detail le contenu de la pile à l'aide du volet de pile ESP et du volet de pile EBP.
- *Practice viewing arguments and local data on the stack pane using Testbug as debuggee*
- Affichez le volet stacktrace pour visualiser la profondeur et la position de l'appel à partir des informations sur la pile.
- Changez le contenu de la pile.
- Explorez en pas-à-pas les APIs pour vérifier si elles retournent une erreur, et si c'est le cas, voir les détails de l'erreur dans le journal.
- *Practice viewing api errors using Testbug as debuggee*
- Utilisez ou **F7** pour exécuter les API précédentes tout en conservant le contrôle. GoBug va se planter si une erreur API se produit - voir Arrêt sur l'erreur de l'API.
- Modifiez les valeurs de registre (ordinaires) ou les flags au moment de l'exécution pour corriger les erreurs ou pour vérifier votre code. Modifiez les registres MMX, les registres à virgule flottante ou certains registres spéciaux (3DNow!, XMM, SSE ou SSE2) au moment de l'exécution pour voir comment ces modifications affectent l'exécution de votre code.
- Visualisez et vérifiez le désassemblage des opcodes produits par votre assembleur ou votre compilateur, visualisez et contrôlez tout code exécutable en tant que mnémoniques assembleur dans le volet de code ou dans l'inspecteur de code, examinez en détail le fonctionnement des mnémoniques en observant les volets.
- Si des symboles sont chargés, affichez une liste des symboles de code et de données pour le debuggee et ses DLLs et créez un inspecteur de code ou un inspecteur de données pour afficher les adresses de code et de données représentées par les symboles.
- Visualisez le debuggee et ses adresses DLLs (les images exécutables) telles que chargées en mémoire : ce sera très différent de l'alternative consistant à regarder le debuggee comme un fichier sur disque avec un visualiseur de fichiers comme PEView par exemple (disponible sur la page assembleur de Wayne J. Radburn) ou similaire. Constituez un inspecteur de code ou un inspecteur de données pour afficher les adresses.
- Parcourez le contexte de la mémoire du debuggee et de la mémoire partagée et visualisez les résultats.
- Recherchez le contexte de la mémoire du debuggee et la mémoire partagée pour des chaînes ou valeurs spécifiques et visualisez les résultats.
- Afficher les informations sur le debuggee à l'exécution.
- Afficher le debuggee et les ressources des DLLs tels chargés en mémoire et, sous réserve du respect d'éventuels droits d'auteur, extraire les ressources utiles des fichiers exécutables.
- Imprimez ou déversez dans un fichier tout ou partie du contenu de l'un des volets.

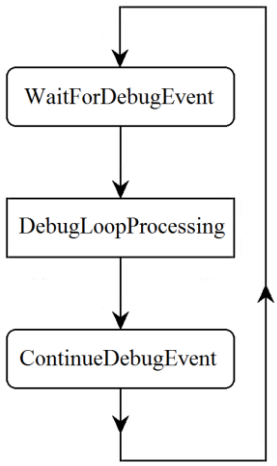
3.1.2 Comment fonctionne GoBug (en tant que débogueur Win32)

3.1.2.1 Débogage dans Win32 - exigences particulières

Un débogueur Win32 doit être spécialement conçu pour surveiller et gérer les événements système, les messages et les actions. En particulier, il doit être capable de surveiller et de signaler les messages vers et depuis le debuggee, car c'est la principale façon par laquelle le système d'exploitation contrôle et communique avec lui. Les messages peuvent être envoyés par le système soit directement au debuggee, soit via la file d'attente de messages (prélevés dans la boucle de message). Lorsque vous attendez un retour de *GetMessage* dans la boucle de message, le débogueur doit être en mesure d'intercepter et de surveiller l'exécution ailleurs dans le debuggee provoquée par les messages envoyés aux procédures de fenêtre. Le débogueur doit également être capable de déboguer des applications multithread, de surveiller les activités, de chaque thread et de visualiser leur interaction. Il doit être capable d'afficher la mémoire virtuelle les zones de pile, les valeurs de registre et d'identifier les zones de mémoire propres au système d'exploitation. Il doit pouvoir suivre l'exécution dans les DLL. Il doit être en mesure de gérer les exceptions causées par le debuggee permettant ainsi à ce dernier de continuer sans se bloquer. Il doit pouvoir signaler les erreurs renvoyées par les API. Enfin, le débogueur doit être capable de fermer le débogueur de manière acceptable.

3.1.2.2 La boucle Debug (du débogueur)

Lorsque vous choisissez un programme à déboguer, GoBug crée un nouveau thread qui démarre le programme en utilisant *CreateProcess*. Ensuite, GoBug entre dans une boucle de débogage. Cette boucle inclut les API Windows *WaitForDebugEvent* et *ContinueDebugEvent*.



L'API *WaitForDebugEvent* renvoie sur l'occurrence de ce qui suit :

(a) après que le processeur a exécuté une instruction du debuggee

(b) quand un message doit être envoyé au debuggee

(c) avant ou après un événement concernant le debuggee (par exemple, la création de fenêtres, un changement de focus, etc.).

Au retour de l'API *WaitForDebugEvent*, GoBug le traite dans "debug loop process" et appelle à nouveau *WaitForDebugEvent*, complétant la boucle.

Les quatre faits qui suivent sont particulièrement intéressants :

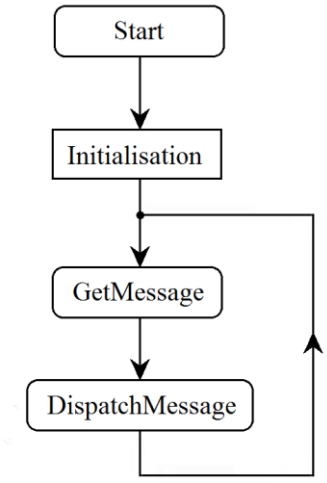
1. Vous pouvez voir d'après ce qui précède qu'une fois qu'une instruction est exécutée par le processeur du débogueur, le débogueur entre dans la boucle de débogage. Lorsque ceci est coché, le débogueur est complètement suspendu, y compris tous les threads du débogué. C'est l'état de GoBug en attendant la saisie de l'utilisateur (F5 à F9).

2. En dépit du fait que tous les threads du debuggee sont suspendus quand le debuggee est dans la boucle de debug, en fait seul un des threads du debuggee a fait entrer le debuggee dans la boucle. C'est le thread pour lequel le processeur a exécuté une instruction pour la dernière fois. En outre, il est possible qu'aucune information à jour ne soit disponible à partir du système à propos des autres threads.

3. Pour mettre dans la boucle de débogage l'instruction suivante exécutée par le processeur pour un thread particulier du debuggee, GoBug doit mettre à 1 le flag de trap dans le contexte du thread dans la boucle. C'est ce que fait GoBug à moins que vous n'appuyiez sur F8 (exécution, hook, partie log) ou sur F9 (exécution en arrière-plan). Si l'une de ces touches est enfoncée, le flag de trap est mis à zéro pour ce thread, de sorte qu'à partir de ce moment-là, les instructions d'un débogueur pour ce thread n'entrent pas dans la boucle. Le résultat de ceci est qu'une fois que vous avez appuyé sur F8 ou F9, il est possible que vous ne puissiez pas retrouver le plein contrôle du débogage sur le thread concerné. Cependant, le contrôle peut généralement être récupéré en définissant un point d'arrêt ou en utilisant la touche de raccourci.

4. Vous pouvez voir d'après ce qui précède qu'il est possible d'avoir différents états "run" pour différents threads de debuggee. Par exemple, le thread principal peut être en cours d'exécution (parce que vous avez appuyé sur F9 lorsque ce thread était dans la boucle), mais tous les autres threads peuvent être examinés en pas-à-pas (flag de trap défini à chaque fois).

3.1.2.3 La Boucle de Message (du debuggee)



Le diagramme ci-contre visualise de fonctionnement d'un programme Windows classique utilisant l'interface utilisateur graphique (GUI). Après son démarrage, le programme initialise la fenêtre principale puis entre dans la boucle des messages. Dans cette boucle, on trouve en premier l'API *GetMessage*. Généralement, cette API reste bouclée sur elle-même tant qu'une touche n'est pas enfoncée ou qu'il n'y a pas de clic de souris. Lorsqu'un tel événement se produit, le process sort de l'API *GetMessage* et s'engage dans *DispatchMessage* dont le rôle est d'envoyer, sous une forme appropriée la pression de touche ou le clic de souris sur la procédure de fenêtre pour traitement. Au retour de la procédure de fenêtre, le programme se retrouve à nouveau dans l'API *GetMessage* et le même cycle reprend.

Il résulte de cette situation que, si vous souhaitez parcourir le code de l'application en pas-à-pas, vous pourrez afficher le code dans la section d'initialisation, puis voir l'appel à *GetMessage*, mais au-delà, vous devrez vous contenter de parcourir la boucle de message. Vous allez donc manquer tout le traitement du message dans la procédure de fenêtre elle-même, puisque tout cela sera traité à votre insu dans l'API *DispatchMessage*.

Pour compliquer cela, vous n'aurez également aucune visibilité sur la façon dont l'application traite les nombreux messages envoyés par le système lui-même (et non envoyés, à ce titre, par la boucle de message) à la procédure de fenêtre.

Pour procéder à un débogage judicieux, il faut trouver un moyen de faire réagir le débogueur aux appels de la fenêtre.

GoBug y parvient grâce au "*break de message pas-à-pas*". Pour ce faire, il détecte quand un message doit passer à la procédure de la fenêtre et définit un point d'arrêt à la fenêtre de procédure prêt à recevoir ce message. Lorsque le point d'arrêt est atteint, GoBug affiche la procédure de la fenêtre dans le volet de code, et vous pouvez choisir d'ignorer le message ou d'évoluer en pas-à-pas dans la procédure de fenêtre.

Vous n'avez pas besoin d'utiliser le "*break de message pas-à-pas*". Vous pouvez l'éteindre si vous le souhaitez et définir vos propres points d'arrêt. Vous pouvez exécuter un message spécifique et vous pouvez spécifier quelle procédure de fenêtre doit recevoir le message avant que le point d'arrêt ne soit déclenché.

3.1.2.4 Comment GoBug surveille les événements dans le debuggee

Vous pouvez voir à partir de ce qui précède que GoBug a besoin de savoir quand les messages sont envoyés au debuggee, que ce soit par le système, par d'autres applications ou par le debuggee lui-même, que ce soit à partir de la boucle des messages ou d'une autre manière. En plus de cela, GoBug a besoin de savoir quels événements se produisent dans le debuggee. Par exemple, si de nouvelles fenêtres ou threads sont créés ou si le debuggee est en train de charger une DLL. Certains de ces événements sont signalés à la boucle de débogage de GoBug par le système en tant qu'*événements de débogage*. Les autres événements et tous les messages sont surveillés par des *hooks* qui consistent en de très petits morceaux de code dans GoBugSpy.dll et qui se chargent dans le contexte de la mémoire du debuggee lorsque le hook est requis pour la constitution du journal. Si vous regardez les zones de mémoire de l'application testée, vous verrez GoBugSpy.dll si elle est là. Il est inhabituel, pour une DLL exécutée par un programme, de charger dans le contexte de la mémoire d'un autre, mais cela est nécessaire pour que le hook fonctionne. GoBugSpy n'est pas chargée si vous exécutez le debuggee en arrière-plan (F9) ou si l'application ne dépend pas de l'interface utilisateur graphique (GUI) du système.

3.1.2.5 Comment GoBug ferme le debuggee

Dans Win32, une application doit souvent être nettoyée avant la fermeture. Cela peut impliquer la fermeture de fichiers, de handles et de mémoire, le déchargement de DLLs ou l'écriture dans la base de registres ou dans un fichier ini. Pour cette raison, à moins que le debuggee ne soit un programme console, GoBug tente de le fermer de manière naturelle en envoyant le message WM_QUIT. Si cela ne parvient pas à fermer le debuggee, GoBug essaiera d'appeler l'API *ExitProcess* au nom du debuggee. Si cela échoue, GoBug terminera son thread de débogage (qui fermera à son tour le thread principal du debuggee). Si cela échoue aussi, après un avertissement, GoBug forcera la fermeture du debuggee en appelant *TerminateProcess*. Evidemment, ce n'est pas une manière optimale de fermer un processus et peut laisser certaines ressources ouvertes.

3.1.2.6 Comment GoBug traite les exceptions

Si une exception se produit dans un programme en cours de débogage, le système envoie d'abord l'exception au débogueur. Cela enjoint au débogueur d'entrer dans la boucle de débogage. GoBug intervient et vous donne les détails de l'exception qui s'est produite. Pour plus de détails, voir la section Exceptions provoquées par le debuggee.

3.1.3 Les fichiers utilisés par GoBug

3.1.3.1 GoBug.exe

C'est l'exécutable principal du débogueur. Vous pouvez lancer l'exécution de deux ou plusieurs copies de GoBug à la fois en faisant une copie de ce fichier avec un nom différent, ou en exécutant GoBug dans un répertoire différent (si vous faites cela, assurez-vous que les DLL y sont également présentes). Il peut être utile d'exécuter une deuxième copie de GoBug si vous voulez tester deux debuggees différents en même temps. Il vous permettra également d'exécuter simultanément deux ou plusieurs copies du debuggee si vous le souhaitez (de manière entièrement séparée et dans leur propre espace d'adressage). Cela peut être utile si vous voulez tester le fonctionnement du debuggee avec des fichiers ou des données partagés, ou si vous voulez comparer en temps réel comment le debuggee réagit à différentes conditions. Notez qu'un nouveau fichier .ini sera créé au nom donné au fichier exe. Chaque copie de GoBug tentera également d'enregistrer la même touche de raccourci au démarrage. Donnez-leur différentes touches de raccourci ou exécutez une copie sans touche de raccourci.

3.1.3.2 GoBugSpy.dll

Ce fichier est injecté dans l'espace d'adressage du debuggee et est utilisé par GoBug pour contenir les hooks nécessaires à la réception des messages destinés au debuggee et aux événements qui surviennent dans le debuggee. Vous ne devez pas changer le nom de ce fichier.

3.1.3.3 GoBug.ini

Ce fichier contient les informations de configuration de GoBug. Il est en format codé et ne peut pas être modifié, sauf en utilisant GoBug lui-même. Les informations qu'il contient incluent la taille, la position, les couleurs et les polices. Il conserve également les noms des debuggee qui ont été débogués dans les sessions précédentes, ainsi que les points d'arrêt, les inspecteurs et les chemins d'accès aux fichiers utilisés dans ces sessions. Si, au démarrage de GoBug, ce dernier ne trouve pas le fichier ini, ou si le fichier ini semble être corrompu, un nouveau sera créé en utilisant une configuration par défaut. C'est ce qui se passe lorsque vous utilisez GoBug pour la première

fois car le fichier GoBug.ini n'est pas livré avec. Si vous changez le nom de GoBug.exe, un nouveau fichier ini sera créé pour correspondre au nouveau nom.

3.1.3.4 GoBug.chm

C'est le fichier d'aide correspondant au présent document, en format de balisage hypertexte compressé.

3.1.3.5 Fichiers système utilisés par GoBug

GoBug s'appuie sur les fichiers système de Windows pour ses fonctionnalités et ne démarrera pas si l'un d'eux est manquant. S'il vous manque HHCTRL.OCX (un fichier système qui montre le present fichier d'aide) exécutez le fichier auto-extractible hhupd.exe (mise à jour de l'aide). Il est disponible sur le site Web de Microsoft.

3.1.4 GoBug en tant que débogueur symbolique

3.1.4.1 Que sont les symboles ?

Les symboles sont les données non locales et les *labels* de code dans votre code source. Un label de *données* se rapporte à une adresse dans la section de données et est constitué lorsque les données sont déclarées. Par exemple,

```
SmallBuffer DD 20h DUP 0
```

crée un buffer de 32 dwords initialisés à zéro, avec le label "SmallBuffer". On peut ensuite se référer à ce buffer par "SmallBuffer" dans votre code source. Un label de code se rapporte à une adresse dans la section de code et peut être spécifié comme suit :

```
Procedure66:
```

ou

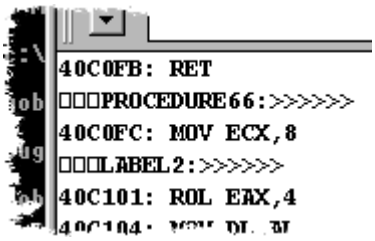
```
Procedure 66 PROC
```

Cela établira le label "Procedure66" dans la section de code. La procédure pourra ensuite être appelée en utilisant le code :

```
CALL Procedure66
```

Un label de donnée ou de code ne doit pas être nécessairement au début de la zone de données ou de code. Vous pouvez diviser les zones de données de sorte que le label ne fasse référence qu'à une partie de celle-ci et qu'un label de code puisse figurer sur n'importe quelle ligne de la section de code.

3.1.4.2 Qu'est-ce qu'un débogueur symbolique ?



Un débogueur symbolique connaît les adresses des symboles et est capable de les afficher dans le désassemblage. Ici, GoBug montre, dans le désassemblage (dans le volet de code), le label de code au début de la procédure, et un second label quelques lignes plus bas. Les références de données dans le désassemblage sont indiquées par un label de données. GoBug peut afficher des symboles Unicode. Les outils "Go" vous permettent d'utiliser le code Unicode et les labels de données dans votre application, et GoBug les affichera correctement. Voir (par exemple) la section Affichage des symboles en Unicode dans le volet de code. Un débogueur symbolique peut également utiliser les labels de code pour permettre à l'utilisateur d'établir des points d'arrêt, et afficher le contenu de la mémoire par référence aux labels de données. Voir la section consacrée aux inspecteurs de code, de données et de symboles.

3.1.4.3 Visualisation des symboles pendant le déboguage

GoBug vous permet de voir les symboles eux-mêmes ainsi que le code et les données représentées par les symboles

- Les symboles de code peuvent être visualisés à l'aide de l'élément de menu "inspect, exe/dll code symbols", et le code à l'adresse du symbole peut être visualisé à l'aide d'un inspecteur de code
- Les symboles de données peuvent être visualisés à l'aide de l'élément de menu "inspect, exe/dll data symbols" et les données à l'adresse du symbole peuvent être visualisées à l'aide d'un inspecteur de données
- Un inspecteur de symboles affichera les données à l'adresse de tous les symboles de données de votre choix

3.1.4.4 Comment GoBug identifie les symboles

L'assembleur place des labels, autres que locaux, dans le fichier objet. L'éditeur de liens, s'il y est invité, triera les labels dans les fichiers objet et les placera dans le fichier exécutable (ou dans un fichier séparé) de manière à ce que GoBug puisse les lire. S'il y a plus d'une source d'information sur les symboles, GoBug utilisera la source selon un ordre prédéterminé. L'ordre de préférence est :

- Symboles COFF incorporés dans le fichier exécutable
- Symboles CodeView incorporés dans le fichier exécutable
- Symboles CodeView dans un fichier .pdb distinct créé ou modifié par l'éditeur de liens
- Symboles COFF d'un fichier .dbg distinct
- Informations de symbole de fichier .map de type Borland ou Microsoft

3.1.4.5 Fichier intégré vs fichier séparé

Les informations de débogage symboliques peuvent être incorporées à l'exécutable ou être entreposées dans un fichier distinct. Il y a des avantages et des inconvénients dans ces deux méthodes :

- Si vous utilisez des informations de débogage incorporées, vous devrez probablement envoyer une version différente de l'exécutable à l'utilisateur (une version sans informations de débogage). Mais si vous utilisez un fichier séparé pour contenir les informations de débogage symboliques, vous n'avez qu'une version de votre exécutable.
- Les informations de débogage incorporées réduisent le nombre de fichiers impliqués et évitent toute confusion entre les exécutables ayant le même nom de fichier (mais une extension différente).
- Le format de fichier pdb peut encore être en développement et il n'est pas officiellement documenté. Les fichiers pdb peuvent être gros et pleins de "boursoufflures".
- Le format de fichier map n'est pas garanti et peut changer avec les versions futures des linkers.
- La taille du dernier symbole dans le fichier map ne sera probablement pas connue.
- Le format de fichier dbg est établi et compact.

À la lumière de ce qui précède, je recommanderais aux programmeurs assembleurs d'utiliser soit une information de débogage intégrée, soit un fichier dbg.

3.1.4.6 Symboles à la fois dans les fichiers exe et les DLL

GoBug recherchera des symbols, à la fois, dans le fichier exécutable principal et dans toutes les DLL chargées par le debuggee. De telles DLL peuvent inclure une DLL qui fait partie du débogueur lui-même ou une DLL système. Une application Windows typique charge plusieurs DLL système au démarrage. À moins que les versions de débogage de ces DLL ne soient chargées sur votre système, vous ne serez pas en mesure d'afficher les symboles de ces DLL système.

3.1.4.7 Le problème des exe et Dll portant le même nom

Une DLL utilisée par votre application peut avoir le même nom que l'exécutable. Par exemple :

```
LotteryForecaster.exe
LotteryForecaster.dll
```

Ce n'est pas un problème pour le débogueur lorsque l'information de débogage est incorporée dans le fichier exécutable lui-même, car il n'y a pas de place pour la confusion quant à la source des informations sur les symboles. Mais lorsque vous utilisez des fichiers de symboles séparés, tels que des fichiers dbg, pdb ou map, vous devez fournir au débogueur un moyen de savoir quel fichier de symboles fait référence à quel exécutable.

Une solution évidente consiste à modifier le nom de la DLL au cours du développement afin que chaque fichier exécutable et son fichier de symbole correspondant aient un nom unique.

Cependant, il existe d'autres solutions selon que vous utilisez des fichiers dbg, pdb ou map :

Fichiers DBG portant le même nom

Lors de l'utilisation de fichiers dbg, les outils vérifient généralement que les fichiers dbg sont placés dans un sous-répertoire "\exe" ou "\dll" du fichier en cours de création (selon qu'il s'agit d'un EXE ou d'une DLL). Ce fait peut ensuite être enregistré dans le répertoire de débogage "MISC" de l'exécutable [*note : il semble que cela n'arrive pas automatiquement lorsque le fichier dbg est créé avec l'API* SplitSymbols*, ni avec le programme rebase, mais cela n'a pas d'importance pour GoBug*] . GoBug recherchera le fichier de symboles dans les sous-répertoires \exe ou \dll s'ils existent. Vous pouvez également placer les fichiers dbg dans les sous-répertoires "\symbols\exe" ou "symbols\dll", étant des sous-répertoires du répertoire dont \bin est un sous-répertoire. Voir le paragraphe consacré aux procédures de recherche de fichiers dbg pour une explication de ceci.

Votre éditeur de liens peut également vous permettre de spécifier un nom pour le fichier dbg, afin que vous puissiez distinguer entre le fichier dbg pour l'exe et le fichier dbg pour la DLL. Le nom correct du fichier dbg est enregistré dans le répertoire de débogage "MISC" de l'exécutable. GoBug cherchera le nom de fichier spécifié.

Fichiers MAP portant le même nom

Au moment de l'édition de liens, vous pouvez normalement spécifier le chemin du fichier map. Assurez-vous qu'il est placé dans un sous-répertoire "\exe" ou "\dll" du fichier en cours, selon qu'il s'agit d'un exe ou d'une DLL. GoBug recherchera dans le sous-répertoire si l'exécutable en cours de chargement est un EXE ou une DLL. Vous pouvez également placer les fichiers map dans les sous-répertoires "\symbols\exe" ou "symbols\dll", étant des sous-répertoires du répertoire dont \bin est un sous-répertoire. Voir le paragraphe consacré aux procédures de recherche de fichier MAP pour en savoir plus.

Fichiers PDB portant le même nom

Au moment de l'édition de liens, vous pouvez spécifier le chemin d'accès et le nom du fichier pdb qui sont alors écrits dans l'exécutable lui-même. GoBug recherche ce chemin d'accès et ce nom et recherche le symbole pdb approprié contenant les symboles de l'exécutable concerné. Vous pouvez également spécifier que le fichier pdb va dans les sous-répertoires \exe ou \dll (ou \symbols\exe ou \symbols\dll selon le cas. Voir les procédures de recherche de fichiers PDB pour toute précision sur ce point).

3.1.4.8 Procédures de recherche pour les fichiers de symboles

Procédure de recherche des fichiers DBG

Lorsque l'on recherche un fichier dbg, ce fichier est recherché si son nom est indiqué dans le répertoire de débogage "MISC". Sinon, GoBug recherche un fichier dbg portant le même nom que l'exécutable. L'algorithme de recherche suivant est utilisé :

- dans le chemin d'accès donné dans le répertoire de débogage "MISC".
- dans le même répertoire que l'EXE ou la DLL en cours de chargement
- dans le sous-répertoire "\exe" ou "\dll" du répertoire de l'exécutable (selon que l'exécutable est un EXE ou une DLL)
- dans le sous-répertoire "\symbols\exe" ou "\symbols\dll" du répertoire parent de l'exécutable. Par exemple, si l'exécutable est c:\windows\bin\LotteryForecaster.exe, le fichier dbg sera recherché dans le sous-répertoire c:\windows\symbols\exe\.

Si le fichier dbg contient des symboles, il contient également le nom du fichier auquel les symboles s'appliquent. GoBug vérifie que cela correspond au fichier correct avant d'utiliser le fichier dbg.

Si le fichier dbg contient des symboles, GoBug vérifie également les date et heure du fichier dbg. En effet, lorsque le fichier dbg est créé, l'exécutable est normalement écrit en même temps, de sorte que les dates et heure de fichier doivent normalement être identiques sinon très proches. Si les dates et heure ne sont pas proches de celles de l'exécutable, le fichier dbg ne sera pas utilisé et GoBug supposera dès lors qu'il s'agit du mauvais fichier.

Procédure de recherche des fichiers MAP

Lorsque vous recherchez un fichier map, GoBug recherche toujours un fichier map portant le même nom que l'exécutable. La séquence de recherche suivante est utilisée :

- dans le même répertoire que l'exe ou la DLL en cours de chargement
- dans le sous-répertoire "\exe" ou "\dll" du répertoire de l'exécutable (selon que l'exécutable est un EXE ou une DLL)
- dans le sous-répertoire "\symbols\exe" ou "\symbols\dll" du répertoire parent de l'exécutable. Par exemple, si l'exécutable est c:\windows\bin\LotteryForecaster.exe, le fichier map sera recherché dans le sous-répertoire c:\windows\symbols\exe\.

GoBug vérifie également l'horodatage du fichier map. C'est parce que le fichier map est fait en même temps que l'exécutable. Si l'horodatage n'est pas proche de celui de l'exécutable, et que c'est l'exécutable principal qui est en cours de chargement et non une DLL, vous serez invité à indiquer si vous voulez utiliser le fichier map (notez que la date et l'heure données seront ajustés pour tenir compte du réglage actuel de l'heure d'été sur votre ordinateur).

Procédure de recherche des fichiers PDB

Lors de la recherche d'un fichier pdb, le nom à rechercher est soit celui donné dans le répertoire de débogage "CODEVIEW", soit dans un fichier dbg dont le nom est donné dans le répertoire de débogage "MISC" de l'exé-

cutable. Si aucune de ces cas ne s'applique, GoBug recherche un fichier pdb portant le même nom que l'exécutable. La séquence de recherche suivante est utilisée :

- dans le chemin d'accès donné dans le répertoire de débogage "CODEVIEW".
- dans le même répertoire que l'EXE ou la DLL en cours de chargement
- dans le sous-répertoire "\exe" ou "\dll" du répertoire de l'exécutable (selon que l'exécutable est un EXE ou une DLL)
- dans le sous-répertoire "\symbols\exe" ou "\symbols\dll" du répertoire parent de l'exécutable. Par exemple, si l'exécutable est c:\windows\bin\LotteryForecaster.exe, le fichier pdb sera recherché dans le sous-répertoire c:\windows\symbols\exe\.

GoBug vérifie également l'horodatage dans le fichier pdb, (ou dans les fichiers ultérieurs un identificateur global unique spécifique à la machine) et les codes « âge » avec ceux contenus dans le répertoire de débogage «CODEVIEW» de l'exécutable. Le code d'âge identifie le nombre de fois que le fichier pdb a été mis à jour. Ces deux valeurs doivent correspondre et si ce n'est pas le cas, le fichier pdb est ignoré.

GoBug n'écrit *jamais* l'ensemble chemin d'accès/nom de fichier du fichier pdb ou le fichier dbg dans l'exécutable lui-même, comme le fait le débogueur VC ++.

3.1.4.9 Que se passe-t-il si les symboles ne sont pas trouvés ?

Si GoBug ne parvient pas à trouver les symboles du fichier exe principal du debuggee, il affichera une boîte de message d'avertissement. GoBug fonctionnera parfaitement bien sans symboles. Cependant, les symboles n'apparaîtront pas lors du désassemblage. Vous ne pourrez pas créer de points d'arrêt ni inspecter des données basées sur des symboles et vous ne serez pas non plus capable de créer un inspecteur de symboles. Si GoBug ne parvient pas à trouver des symboles dans une DLL, que ce soit les propres DLL du debuggee ou une DLL système, il n'y aura pas de boîte de message d'erreur. Si vous voulez vérifier quels symboles de fichiers chargés ont été trouvés, cliquez sur l'élément de menu "inspect, information about the debuggee" ("inspecter, informations sur le débogueur").

3.1.4.10 Ajout d'informations sur les symboles incorporés avec divers liens

En utilisant GoLink, utilisez le commutateur "/debug coff". Le format est COFF.

Si vous utilisez l'éditeur de liens ALINK d'Anthony Williams, utilisez le commutateur "-debug". Le format est de type CodeView.

Avec les anciens linkers Microsoft, vous pouvez utiliser le commutateur "/DEBUG: FULL" puis "/DEBUGTYPE: COFF" ou "/DEBUGTYPE: CV" pour ajouter des informations respectivement de débogage COFF ou CodeView incorporées au fichier exécutable.

Le dernier éditeur de liens Microsoft n'intègre pas les informations de symbole CodeView. Au lieu de cela, les informations symboliques sont placées dans un fichier de base de données de programme (pdb) distinct qui peut être lu par GoBug. Voir le paragraphe Création d'un fichier pdb.

Cependant, le commutateur /DEBUGTYPE: COFF intègre toujours les informations COFF dans l'exécutable du dernier éditeur de liens Microsoft, tout comme /DEBUGTYPE: BOTH. Si vous le souhaitez, vous pouvez supprimer les symboles de l'exécutable par la création d'un fichier DBG, et GoBug lira les symboles du fichier DBG.

Si vous utilisez TLINK de Borland, utilisez le commutateur /v pour ajouter des informations de débogage à l'exécutable. GoBug peut lire les symboles de format FB09 ou FB0A qui sont ensuite émis. Les versions ultérieures de TLINK peuvent utiliser de nouveaux formats. Je vous remercie par avance de m'envoyer le fichier exécutable si vous rencontrez des problèmes.

3.1.4.11 Création d'un fichier DBG

En utilisant l'éditeur de liens GoLink, utilisez le commutateur "/debug dbg". Cela permettra de produire un fichier dbg dans un répertoire \exe ou \dll selon le cas.
Une autre façon de créer un fichier dbg est de constituer l'exécutable avec les informations COFF intégrées, puis d'exécuter rebase.exe de Microsoft avec le commutateur approprié (essayez la ligne de commande "REBASE -x Drive:\Path -b 0x400000 FileName.Ext" où FileName.Ext est l'exécutable concerné, rebase crée un fichier dbg dans un sous-répertoire dans Drive:\Path appelé EXE ou DLL en fonction de l'extension du nom de fichier). Vous pouvez également utiliser l'API *SplitSymbols* qui semble effectuer le même travail.

3.1.4.12 Création d'un fichier MAP

Avec le linker Microsoft, vous pouvez utiliser le commutateur "/MAP" pour créer un fichier map qui peut être lu par GoBug. Si vous utilisez TLINK de Borland, utilisez le commutateur /m pour faire de même. L'éditeur de liens ALINK d'Anthony Williams crée également un fichier map en utilisant le commutateur "-m", mais c'est

dans un format que GoBug ne peut pas utiliser. Il peut cependant être utilisé comme méthode manuelle d'identification des adresses de symboles de données et de code (où GoBug s'exécute sans symboles chargés).

3.1.4.13 Création d'un fichier PDB

Les derniers linkers Microsoft créent un fichier de base de données de programme (pdb) lors de l'édition de liens si l'option /DEBUG ou l'option /DEBUG: FULL est sélectionnée. Le fichier pdb contient des informations sur la création de l'exécutable et contient également les informations de symbole dans le dernier format CodeView. L'exécutable contient un chemin d'accès et un nom de fichier pour le fichier pdb sur la machine locale, avec un code d'identification, de sorte que le fichier pdb correct peut être localisé. Ni le format du fichier pdb lui-même ni le dernier format CodeView ne sont documentés. Cependant, GoBug lit ces fichiers tels qu'ils sont actuellement connus, c'est-à-dire qu'il lit les formats "JG" et "DS" (également nommés "RSDS").

Si les commutateurs ci-dessus sont utilisés, l'utilitaire REBASE crée maintenant un fichier dbg qui contient simplement un chemin d'accès et un identificateur du fichier pdb.

3.1.5 Autres techniques à essayer (au lieu de déboguer)

L'essentiel du contenu de cette section figure dans l'annexe H du volume 1 de l'assembleur GoAsm auquel il convient de se referrer pour ce faire. Le paragraphe ci-dessous est celui qui ne figure pas dans l'annexe H.

3.1.5.1 Recherche de problèmes de correction ou de liaison lorsque la ligne de code source n'est pas connue

Parfois, votre éditeur de liens signale un problème de correction ou d'édition de liens sans en identifier la position exacte dans votre code source. Ces problèmes peuvent être plus difficiles à trouver, et c'est l'une des raisons pour lesquelles le codage incremental est recommandé, lequel consiste à tester chaque nouvelle partie du code tel qu'il est écrit. Si vous avez besoin de trouver un problème de ce type et que vous ne pouvez pas identifier la source du problème, essayez d'insérer un mnémonique NOP à différents endroits de votre code source (mais ne mettez qu'un NOP à la fois). L'adresse de l'instruction après le NOP se verra ainsi augmentée de 1 octet. Si l'adresse d'erreur change maintenant, vous savez que l'erreur est inférieure au NOP. Vous pouvez également mettre en commentaire une ligne contenant une relocalisation (adresse mémoire d'un symbole) qui aura pour effet de réduire l'adresse d'erreur de quelques octets si elle se trouve en dessous de la ligne. Vous pouvez faire une estimation approximative de la position de l'erreur dans votre code source en comparant l'adresse supérieure ou le nombre total de relocalisations et comparer cela avec la valeur donnée dans le message d'erreur. Souvenez-vous que si vous avez plusieurs fichiers OBJ que l'éditeur de liens les liera probablement dans l'ordre dans lequel ils sont lus. Vérifiez la séquence des fichiers OBJ présentés à l'éditeur de liens pour avoir une idée du fichier dans lequel le problème se trouve.

3.1.6 Le programme de test Testbug

3.1.6.1 Qu'est-ce que Testbug ?

Testbug est un programme livré avec GoBug. Il peut être utilisé pour pratiquer le débogage et tester GoBug. Il comporte également de nombreux exemples de code et visant à illustrer l'utilisation de l'assembleur et de la programmation Win32. Voici une brève description de ce que fait Testbug.

3.1.6.2 Les fichiers utilisés par Testbug

L'exécutable principal est Testbug.Exe et les DLLs sont respectivement Testbug1.dll, Testbug2.dll et Testbug3.dll. Signalons également qu'il existe un document complet sur le contenu de Testbug.exe (Testbug.chm sur le site **http://www.GoDevTool.com**). Tous les exécutables ont leurs symboles incorporés au format Coff. Testbug1.dll et Testbug2.dll sont chargées lorsque le programme démarre. Testbug3.dll est chargée au moment de l'exécution lors de l'essai du lien d'exécution et des tests de code de démarrage.

3.1.6.3 Les tests de rapidité d'exécution effectués par Testbug

Ces tests comparent la vitesse entre diverses API ayant des fonctions semblables et comparent également la vitesse de certaines API avec leurs homologues écrites en assembleur. Les sujets traités sont la vitesse d'adressage de la pile, l'optimisation de LOOP et REP, la vitesse comparée FPU/CPU, la vitesse d'exécution portant sur des données dans la section code, l'optimisation des branchements avec l'Intel Pentium P4 et plus, l'affichage de nombres à l'écran et, enfin, la vitesse d'exécution des API de gestion de mémoire. Pour plus d'informations, voir la description des fonctionnalités de Testbug.exe dans le volume 3 de la série décrivant les outils assembleur Go.

3.1.6.4 Testbug : tests "use of Win32"

Ces tests donnent des exemples pratiques de l'utilisation de Win32 dans un contexte assembleur :

- les different usages de la pile
- création et utilisation de DLL
- notifications temporaires à l'écran
- découpage de fenêtre par le contexte de périphérique
- découpage de fenêtre avec les API de fenêtre
- utilisation des API de gestion de la mémoire
- création de ressources dans différentes versions linguistiques
- utilisation du Microsoft Layer for Unicode
- simulation d'une boîte de dialogue en utilisant une fenêtre ordinaire
- utilisation de différents types de WndProc.

Le fichier Testbug.chm du site **http://www.GoDevTool.com** fournit tous les details utiles à cet égard.

3.1.6.5 Testbug : tests "use of GoBug"

Ces tests montrent l'action de GoBug en temps réel. Voir, à ce sujet, le paragraphe Utilisation de Testbug pour pratiquer le débogage et utiliser GoBug.

3.1.6.6 Testbug : tests "use of mnemonics"

Ces tests analysent l'utilisation de divers mnémoniques :

- instructions NEG et NOT
- taille du code produit par divers mnémoniques
- utilisation des mnémoniques de contrôle de bits
- utilisation des flags
- utilisation de REP SCAS, IDIV et IMUL
- codage BCD
- différents types de JMP et CALLs
- utilisation des instructions à virgule flottante, des instructions MMX, 3DNow !, XMM, SSE et SSE2
- contrôle à virgule flottante SIMD
- calculatrice hexadécimale en temps réel.
- Le fichier Testbug.chm du site **http://www.GoDevTool.com** fournit tous les details utiles à cet égard.

Utilisation de Testbug pour pratiquer le débogage et utiliser GoBug

Testbug est décrit dans le volume 3 de cette série. Ce document explore les thèmes suivants :

- Saut conditionnel et utilisation de tests de flags (un bon test pour essayer le mode pas-à-pas)
- Erreurs Api (montre comment vous pouvez voir les erreurs API survenant lors de l'exécution)
- récursion WndProc (illustre les breaks de message pas-à-pas et les messages récursifs)
- Tests de pile (montre la visualisation des arguments et le stockage des données locales sur la pile)
- Attente d'un retour de la part d'un CALL
- Boucles infinies (un bon test pour tester le raccourci clavier ou le contrôle par feux tricolores)
- Débogage des threads (montre ce qui se passe lorsque le debuggee a plus d'un thread)
- Exceptions (montre ce qui se passe lorsque le debuggee provoque une exception)
- Messages (montre comment vous pouvez voir les messages envoyés depuis et vers le debuggee)

3.2 Structure de GoBug

3.2.1 Le code, les registres principaux et les volets de pile

3.2.1.1 Ce que montrent les volets

Les volets donnent un "instantané" du debuggee à un moment du processus d'exécution. Si le debuggee est actuellement dans la boucle de débogage, il sera dans un état suspendu, et les volets afficheront cet état courant. L'état inclut les valeurs de registres en cours (volet de registre), le contenu de la pile en cours (le volet de pile ESP et le volet de pile EBP) et la dernière ligne de code exécutée (volet de code). Les volets ont un arrière-plan blanc si le thread examiné est actuellement dans la boucle de débogage ou s'il retournera dans la boucle de débogage lorsque le système reviendra d'une API.

Sous Win32, c'est le système, et non le processeur, qui détient les valeurs de registre. Ceci, parce que le processeur doit être commuté pour passer l'exécution d'un programme à l'autre. Les valeurs de registre sont conservées dans n'importe quelle zone de mémoire conservée par le système pour chaque processus appelé "contexte de registre". Quand c'est le tour d'un thread d'obtenir le temps du processeur, son contexte de registre sera donné au processeur. Dans la boucle de débogage, le contexte de registre est lu dans la boucle de débogage à l'aide de

l'API *GetThreadContext*. Cette API donne uniquement des résultats valides dans la boucle de débogage (lorsque le processus est suspendu).

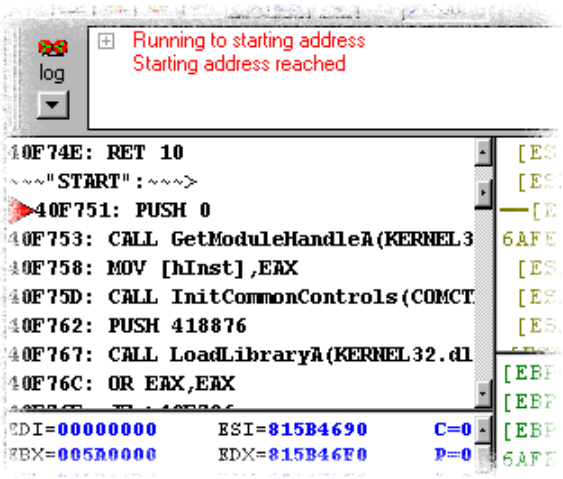
En ce qui concerne la pile, au démarrage, chaque programme reçoit une petite quantité de mémoire physique pour sa pile. À mesure que les besoins de la pile augmentent, le système alloue plus de mémoire. La zone de mémoire utilisée pour la pile est contenue dans le "contexte de mémoire" du processus.

3.2.1.2 Cas de volets grisés

Si les volets sont grisés, cela signifie que le thread qui y est affiché n'est pas actuellement dans la boucle de débogage, probablement parce qu'il est en cours d'exécution. Le fait que des volets soient grisés vous indique que les informations qui y sont affichées ne rendent pas nécessairement compte avec précision de l'état actuel du thread.

Notez que si vous avez appuyé sur F6 pour tracer une procédure et que le debuggee n'est pas encore retourné à partir de cette procédure (qu'il s'agisse d'une API ou d'une procédure dans son propre code), les volets ne sont pas grisés. C'est parce que GoBug attend le retour de la procédure.

3.2.1.3 Le volet de code

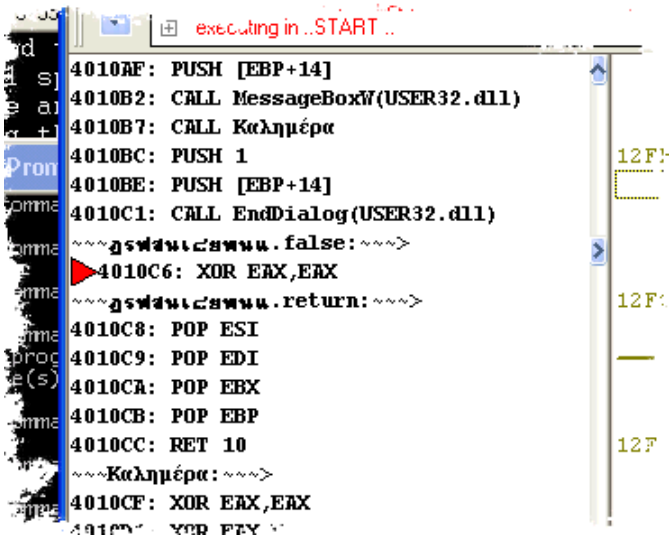


Le volet de code affiche l'emplacement actuel de l'instruction (pointé par la flèche rouge), le désassemblage du code à cet endroit et les lignes d'instructions à proximité. GoBug est un débogueur symbolique. Cela permet d'afficher les noms des labels de code. "START" apparaît ici, montrant l'adresse de démarrage du programme. Les symboles dans la section de données sont également affichés. A titre d'exemple, vous pouvez voir `MOV [hInst],EAX` à la 4ème ligne. Il s'agit d'une instruction de déplacement du registre EAX dans le symbole de section de données nommé "hInst" (les crochets indiquent une référence à un emplacement mémoire). Dans cette illustration, il y a également 3 appels aux API Windows. Le nom de l'API est donné en premier suivi de celui de la DLL Windows qui l'héberge.

Tous les nombres affichés dans le volet de code sont en hexadécimal. Ceux de la colonne de gauche correspondent à l'adresse d'instruction chargée dans le contexte de mémoire du debuggee par Windows.

Utilisez la barre de défilement sur le côté droit pour faire défiler vers le haut ou vers le bas la section de code que vous visualisez. Dans la barre de défilement il y a un bouton avec une petite flèche droite. C'est le bouton de défilement. Déplacez-le pour faire défiler vers le haut ou vers le bas le volet de code (une fenêtre apparaîtra montrant l'adresse courante affichée). Vous pouvez également cliquer dessus pour développer le le volet de code afin que vous puissiez voir tout le texte qu'il contient. Si le volet de code a le focus, vous pouvez utiliser la touche fléchée droite pour le développer et la touche fléchée gauche pour le rétracter.

3.2.1.4 Affichage des symboles en Unicode dans le volet de code



Les outils "Go" vous permettent de manipuler des labels de code et de données en Unicode dans vos applications et ceux-ci sont affichés correctement par GoBug. Voici un exemple d'affichage du volet de code où certains symboles avec des caractères non-romains sont utilisés.

3.2.1.5 Visualisation de différentes sections de code dans le volet de code

Déplacez-vous vers une section de code différente en utilisant l'élément de menu "inspect, codepane to view code at ..", ce qui vous permet de choisir l'adresse à afficher (ou le symbole de code à afficher si chargé). À l'aide du volet de code, vous pouvez uniquement afficher les sections de code du debuggee en tant que tel (y compris les DLL chargées). Si vous souhaitez afficher le code 32 bits dans toute autre zone de mémoire, vous devez créer un inspecteur de code.

3.2.1.6 Visualisation des opcodes dans le volet de code

En cochant l'option de menu "view, codepane opcodes mouseover" (ou la même chose dans le menu contextuel issu du clic droit dans le volet de code), les opcodes sont affichés au passage de la souris au-dessus du volet de code plutôt qu'au désassemblage.

3.2.1.7 Valeurs immédiates négatives dans le volet de code

GoBug n'affiche normalement pas les valeurs immédiates en tant que nombres signés. Cela signifie que, si vous codez `MOV EAX, 88000000h`, c'est ce qui apparaîtra dans le code sur le désassemblage. Ceci en dépit du fait que le bit le plus élevé de ce nombre est à 1 et qu'il peut être considéré comme un nombre (signé) négatif (valeur inférieure à 78000000h). Cependant, dans certains cas, GoBug supposera qu'une valeur signée a été prévue par le codeur et qu'il peut donc afficher le nombre comme étant négatif. Cela se produit si la forme courte de codage est trouvée lors du désassemblage, c'est-à-dire l'opcode 83, qui utilise seulement 3 octets au lieu des habituels 6 octets. Ceci est normalement produit si vous codez `ADC`, `ADD`, `AND`, `CMP`, `OR`, `SBB`, `SUB` et `XOR` en utilisant des registres 32 bits et où l'opérande est un nombre compris entre –80h et +7Fh. Notez cependant que tous les assembleurs ne le feront pas. De la même manière, le fait de pratiquer un PUSH sur une valeur comprise entre –80h et +7Fh peut être fait avec deux octets d'opcode en utilisant l'opcode 6A, et GoBug les traite aussi, comme des nombres signés.

3.2.1.8 La flèche EIP

Cette flèche pointe vers l'instruction en cours *avant* qu'elle soit exécutée. Appuyer sur F5 exécutera cette instruction et permettra de passer à la suivante. Les sauts et sauts conditionnels s'afficheront comme une flèche vers le haut ou vers le bas (il y aura également un changement de couleur). Si vous changez les flags alors que le pointeur est sur un saut conditionnel, la flèche changera pour s'adapter aux nouveaux flags. La flèche peut également changer de direction si vous modifiez une valeur de registre testée par une instruction, par exemple en changeant ECX à zéro sur une instruction LOOP.

Si vous pouvez voir le volet de code avec un fond blanc, mais que vous ne pouvez pas voir la flèche EIP, c'est parce que vous visualisez une zone de code à l'écart de l'instruction en cours. Utilisez "inspect, codepane to view at .." pour restaurer la vue du volet de code à l'EIP en cours. Notez que dans une application multithread, chaque thread aura sa propre valeur EIP. Choisissez l'EIP correct pour la section de code effectivement dans la boucle de débogage, indiquée par le bouton de thread.

Si la flèche est intermittente mais pointe vers une instruction CALL, cela indique que le CALL est en cours d'exécution mais que ledit CALL n'est pas encore retourné. Le CALL peut effectuer un long processus, il peut

être en attente d'un événement ou il peut être coincé dans une boucle. Voir le pragraphe traitant de l'action si un thread n'est pas retourné d'un CALL.

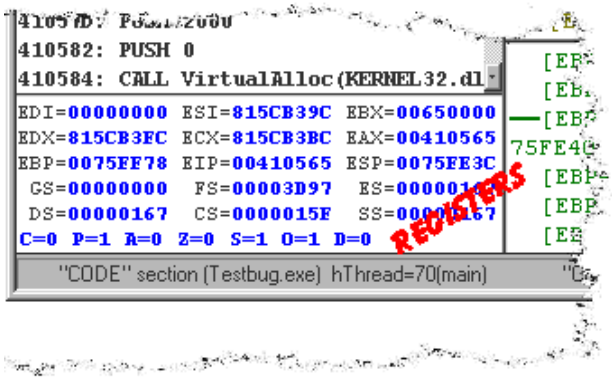
3.2.1.9 Répétition d'un volet de code dans une nouvelle session

Si vous démarrez une nouvelle session avec le même debuggee, les dernières vues du volet de code (que ce soit par adresse de code ou par nom de procédure) apparaîtront dans le menu "inspect, codepane to view at ..". Voir sessions

3.2.1.10 Menu de contexte dans un volet de code

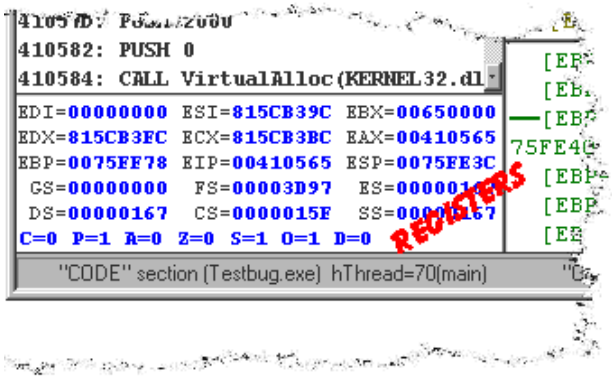
Un clic droit dans le volet de code donne un menu contextuel à partir duquel vous pouvez facilement choisir différentes options, y compris la définition d'un point d'arrêt et le traitement de l'apparence du volet de code.

3.2.1.11 Le volet de registres



Le volet de registres affiche les valeurs courantes contenues dans les registres ordinaires. Ce sont toutes des valeurs hexadécimales. Dans le cas des registres de segments (DS, CS, SS, FS, GS), bien qu'il s'agisse de registres à 32 bits, seuls les 16 premiers bits contiennent des valeurs valides, le reste des bits étant inutilisé. Les flags sont également affichés dans le volet de registre. Ces flags sont C (carry), P (parité), A =, Z (zéro), S (signe), O (débordement) et D (direction).

3.2.1.12 Changement du contenu des registres et des flags



Lorsque vous êtes dans la boucle de débogage, vous pouvez changer la valeur des registres. Pour ce faire, faites un clic gauche sur le registre, effacez les chiffres que vous voulez modifier et entrez la nouvelle valeur (appuyez sur Entrée lorsque vous avez terminé ou sur ESC pour annuler). Vous pouvez également changer les flags. Pour ce faire, faites un clic gauche sur le flag pour le changer.

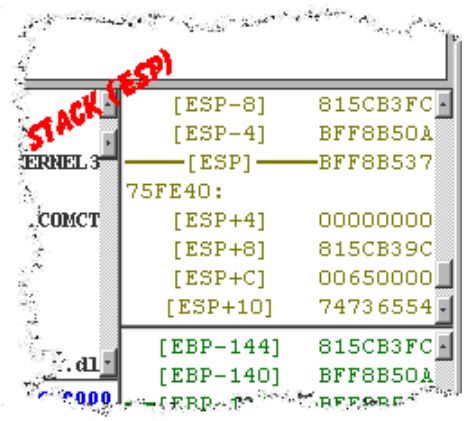
Lorsque le volet de registres a le focus, au lieu de cliquer sur la souris, vous pouvez appuyer deux fois sur Entrée pour modifier le registre ou le flag qui a le rectangle de focus. Si le registre ou le flag est déjà en surbrillance, il suffit d'appuyer une fois sur Entrée.

Lorsque vous parcourez le code, les registres et les flags dont la valeur a changé depuis la dernière instruction apparaissent dans une couleur différente.

3.2.1.13 Menu contextuel dans le volet de registres

Un clic droit dans le registre donne un menu contextuel à partir duquel vous pouvez facilement choisir différentes options, y compris modifier un registre ou inspecter une valeur contenue dans celui-ci.

3.2.1.14 Le volet de pile (ESP)

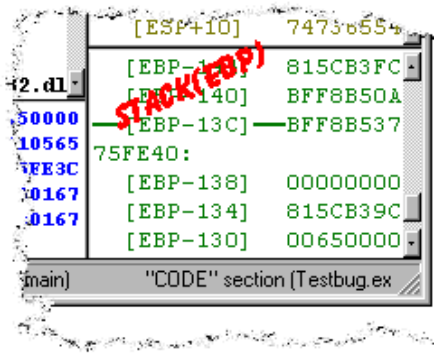


Le volet de pile affiche le contenu de la pile pour le thread qui se trouve dans la boucle de débogage. Au début d'une session de débogage, le volet ESP affichera la pile à la position du pointeur de pile (ceci est indiqué par une ligne). Cet endroit est où n'importe quelle instruction PUSH ou POP utilisera la pile. Au fur et à mesure que vous parcourez le code, tout contenu de pile dont la valeur a changé depuis la dernière instruction apparaîtra dans une couleur différente.

Chaque ligne de l'affichage de la pile représente une position de pile (qui, dans Win32, correspond à un Dword). Le côté gauche de l'affichage de la pile montre le décalage de ESP pour la ligne affichée. L'adresse de pile réelle apparaît toutes les 4 lignes. Comparez ces valeurs avec celle de ESP affichée dans le volet de registres pour comprendre l'affichage de la pile. La colonne de droite affiche le contenu réel de la pile. Notez que lorsque vous effectuez un appel d'API ou un appel à une procédure, vous constaterez probablement que plusieurs modifications se produisent dans la pile dans de nombreux endroits qui sont hors de vue. Faites défiler vers le haut et vers le bas pour afficher le reste de la pile.

Voir aussi le volet de traçage de la pile.

3.2.1.15 Le volet de pile (EBP)



Le volet de pile EBP peut être utilisé pour scruter les données locales conservées sur la pile dans une trame de pile dans une procédure. Encore une fois, la ligne montre l'adresse contenue dans le registre ESP. Le codage qui suit montre comment utiliser le volet de pile EBP en pratique (exemple tiré d'une procédure de dialogue) :

```
1. DlgProc:
2. PUSH EBP
3. MOV EBP, ESP
4. SUB ESP, 40h
;
;main dialog procedure code goes here
;
MOV ESP, EBP
POP EBP
RET 10h
```

Ce code montre l'établissement d'une trame de pile pour une procédure de dialogue. Il a deux tâches à faire. Premièrement, il doit faire en sorte que EBP soit utilisé pour traiter les arguments envoyés par l'appelant. Deuxièmement, il doit mettre en place une zone sur la pile où les données locales peuvent être stockées pendant la procédure. La ligne 3 configure EBP qui va permettre de traiter les arguments. Après que cette instruction est exécutée, les arguments envoyés par le système peuvent être adressés comme suit :

hDlg à [EBP+8], uMsg à [EBP+0Ch], wParam à [EBP+10h] et lParam à [EBP+14h].

Il en résulte que vous pouvez les observer en consultant, tout simplement, le volet de pile EBP.

La ligne 4 met l'emplacement des données locales sur la pile en laissant assez d'espace pour 16 mots (c'est-à-dire 16×4 octets). Une fois cette instruction exécutée, les données locales sont adressables en tant que `[EBP-4]` à `[EBP-40h]`. Tout PUSH supplémentaire n'empiétera pas sur cette zone car ESP est maintenant à une valeur inférieure. Encore une fois, vous serez en mesure d'afficher les données locales en utilisant le volet de pile EBP.

Lorsque la trame de la pile est fermée, ESP et EBP sont restaurés aux valeurs précédentes et l'instruction RET 10h déplace ESP de 4 mots avant l'instruction RET, de sorte que le code retourne correctement à l'appelant.

Voir aussi le volet de traçage de la pile.

3.2.1.16 Changement du contenu de la pile

Lorsque vous êtes dans la boucle de débogage, vous pouvez modifier le contenu de la pile. C'est utile si vous voulez essayer de passer différents paramètres à une API. Pour ce faire, faites un clic gauche sur la position de la pile que vous voulez modifier. Effacez les chiffres correspondants et entrez la nouvelle valeur (appuyez sur Entrée lorsque vous avez terminé ou sur ESC pour annuler).

Lorsque le volet de pile a le focus, au lieu de cliquer sur la souris, vous pouvez appuyer deux fois sur Entrée pour changer la position de la pile qui a le rectangle de focus. Si le registre ou le flag est déjà en surbrillance, il suffit d'appuyer une fois sur Entrée.

3.2.1.17 Menu contextuel dans les volets de la pile

Un clic droit dans le volet de pile ouvre un menu contextuel à partir duquel vous pouvez facilement choisir diverses options, y compris modifier la pile et inspecter une valeur contenue dans celle-ci.

3.2.1.18 Rafraichissement des volets

Le contenu des volets est actualisé immédiatement après chaque étape de pas_à-pas. Si vous souhaitez actualiser les volets à un autre moment, vous pouvez cliquer sur le bouton de thread approprié href = "Multi.htm # tb"> ou utiliser l'élément de menu "inspect, refresh all panes/inspectors" ("inspecter, rafraîchir tous les volets/inspecteurs").

3.2.1.19 Changer le nombre et la position des volets

Utilisez les boutons du volet de la barre d'outils pour modifier le nombre et la position des volets principaux. Les boutons (en partant de la gauche) correspondent au volet de code, au volet de registre, au volet de pile ESP, au volet de pile EBP et au volet de journal de bord (oui, c'est supposé être un journal!). Pour plus d'informations sur le volet de journal de bord, cliquez ici. Pour supprimer un volet, éteignez son bouton. Pour le restaurer, rallumez-le. L'ordre dans lequel vous activez les volets détermine leur position, car ils remplissent la fenêtre principale dans l'ordre suivant : en haut à gauche, en bas à gauche, en haut à droite et en bas à droite.

3.2.1.20 Modification de la taille des volets

Chaque volet peut voir sa taille modifiée de la manière habituelle à savoir, en faisant glisser son bord.

3.2.1.21 Restauration de la taille, du nombre et de la position des volets

Vous pouvez restaurer la taille, le nombre et la position des volets principaux (code, registre et pile) par défaut en cliquant sur l'élément de menu "view, panes as default".

3.2.1.22 Vidage des volets

Assurez-vous que le volet que vous souhaitez supprimer est visible. Faites-le défiler pour qu'il indique le point de départ du matériau que vous souhaitez vider. Cliquez sur l'élément de menu "file, dump" ("fichier, vidage"). Cliquez sur le volet approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug en proposera un). Choisissez un chemin d'accès pour que le fichier soit écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguez cet exe particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains). Si vous videz le volet de code ou de pile, vous pouvez changer le point de départ et la taille de la sauvegarde. Si vous videz le volet de registre, il ne s'agit que d'une seule page. Le fichier créé sur cette image est du texte brut avec des retours chariot et des sauts de ligne mais pas de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur défini sur une police de largeur fixe (par opposition à une police proportionnelle).

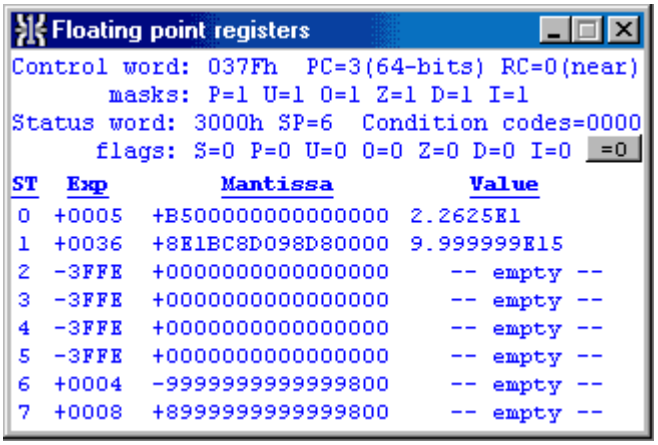
3.2.1.23 Impression des volets

Assurez-vous que le volet que vous souhaitez imprimer est visible. Faites défiler ce volet pour afficher le point de départ du matériau que vous souhaitez imprimer. Cliquez sur l'élément de menu "file, print" ("fichier, imprimer"). Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Cliquez sur le volet approprié qui apparaîtra dans l'élément de menu. Si vous imprimez le volet de code ou de pile, vous pouvez modifier le point de départ et la taille de l'impression. Si vous imprimez le volet de registres, il ne s'agit que d'une seule page. Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.2.2 Les volets de registres spéciaux (MMX, 3DNow!, XMM, SSE et SSE2)

Voir Testbug pour les tests de ces volets de registre spéciaux

3.2.2.1 Le volet de registres virgule-flottante



C'est le volet affichant les valeurs courante dans la FPU (unité de calcul en virgule flottante). Pour afficher le volet de registres à virgule flottante, cliquez sur le bouton de la barre d'outils "MMX/FPU", puis choisissez le bouton du milieu dans la barre d'outils flottante.

Les quatre premières lignes de l'affichage du volet affichent le mot de contrôle, les bits de précision et d'arrondi, les masques d'exception, le mot d'état, le pointeur de pile et les codes de condition, ainsi que les flags d'exception. Sous ces informations, vous pouvez observer le contenu des registres à virgule flottante. La colonne de gauche montre le registre, puis l'exposant, le signe et la mantisse, suivis de la valeur du nombre réel correspondant. Enfin, pour revenir à la quatrième ligne sur le côté droit, il y a le bouton de mise à zéro du flag d'exception.

3.2.2.2 Les instructions et registres à virgule flottante

Les instructions à virgule flottante sont des mnémoniques qui utilisent les *registres à virgule flottante*. Ces registres sont parfois appelés registres FPU (unité de calcul en virgule flottante). C'est parce qu'avant, l'introduction du processeur Intel 486, la FPU était dans une puce séparée appelée le co-processeur mathématique. Ces puces étaient nommées, selon le processeur principal, 8087, 287 ou 387 et implantées sur la carte mère, le plus souvent en option. De nos jours les registres à virgule flottante sont incorporés dans la puce de processeur principale. Les registres à virgule flottante permettent des calculs mathématiques utilisant des nombres réels, ce qui signifie que des nombres très grands ou très petits peuvent être manipulés, dépassant la limite de 32 bits des registres ordinaires. Ceci est réalisé au moyen d'une architecture 80 bits, qui peut être lue ou recevoir des données en utilisant une variété de types de données. Il existe à cet effet une batterie d'instructions spécifiques dont les mnémoniques commencent tous par la lettre "F". Les documents d'Intel et d'AMD décrivent, par le détail, ces instructions et donnent de nombreux exemples d'application.

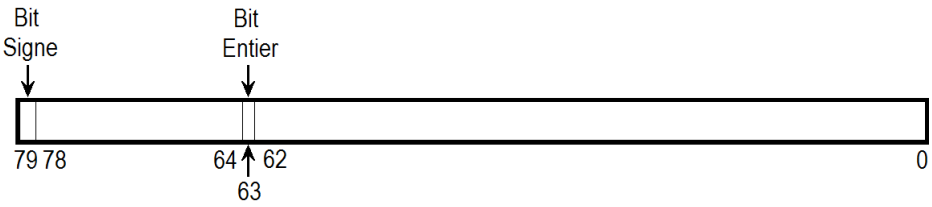
Il y a huit registres en virgule-flottants en tout.

3.2.2.3 Accès aux registres virgule-flottante

Les registres à virgule flottante sont conçus de telle sorte qu'ils sont généralement écrits ou lus par des instructions qui ont une action "push" ou "pop" conjointe. Celles-ci utilisent les registres à virgule flottante de la même manière que les instructions PUSH et POP utilisent la mémoire de pile. Comme il y a huit registres à virgule flottante, huit commandes "push" successives rempliront tous les registres et une neuvième "push" ne fonctionnera pas. Lorsque vous utilisez une commande "pop", vous retirez de la pile la dernière valeur insérée. Les plus courantes de ces instructions sont FLD (chargement), FILD (conversion en un entier, mémorisation et POP), FSTP (mémorisation et POP), FISTP (conversion en un entier et POP).

3.2.2.4 Signe, mantisse et exposant

Les nombres réels, tels que stockés dans les registres à virgule flottante de 80 bits, se décomposent en 3 parties distinctes : signe, mantisse et exposant (un exposant binaire). La manière exacte dont ces valeurs sont insérées dans la pile à virgule flottante et extraites de la même pile dépend du type de données utilisé. Consultez les instructions de l'assembleur relatives à ces différents types. Quel que soit le type de données utilisé, lorsque le registre est chargé avec un nombre, le processeur le convertit toujours au format 80 bits en "précision étendue" ("extended precision"). Et lors de la suppression d'un nombre du registre, le processeur le convertira du format 80 bits dans le type de données souhaité. Par conséquent, les types de données n'affectent pas la façon dont les registres à virgule flottante stockent les nombres. C'est toujours dans le format 80 bits, c'est-à-dire :



bit 79 = signe du nombre réel

bits 64 à 78 = exposant binaire (avec un offset – voir ci-dessous)

bit 63 = partie entière de la mantisse. Toujours à 1. La valeur 0 est utilisée pour caractériser un état spécial (de-normal, etc.)

bits 0 à 63 = contiennent mantisse du nombre (en binaire)

Par conséquent, dans son affichage du contenu des registres à 80 bits, GoBug ne fait pas de distinction entre les types de données, mais affiche simplement le contenu des registres à 80 bits dans ces trois composantes. GoBug, cependant, ajuste la valeur de l'exposant binaire pour tenir compte de l'offset de 16 382 décimal introduit dans l'expression de l'exposant (lequel permet tout simplement de préserver une continuité des valeurs d'exposant lorsque ces derniers passent du négatif au positif et inversement). En vertu de ce principe, si GoBug affiche un exposant binaire nul, la valeur portée dans les bits 64 à 78 est en fait de 16 382 (valeur décimale correspondant à 3FFEh).

Si l'exposant binaire a la valeur 1, le nombre réel est simplement la mantisse sous forme décimale. Si l'exposant binaire est 2, le nombre réel est la mantisse multipliée par 2 sous forme décimale. Si l'exposant binaire est 3, le nombre réel est la mantisse multipliée par 4 sous forme décimale. Il en résulte que le nombre réel représenté par le contenu du registre à virgule flottante peut s'exprimer par la relation :

$$\text{mantisse} \times 2^{\text{exposant-1}}$$

Cela peut être testé en entrant les chiffres suivants pour l'exposant et la mantisse respectivement :

```
exposant = 1 mantisse=80000000 00000000 valeur=1E0
exposant = 2 mantisse=80000000 00000000 valeur=2E0
exposant = 3 mantisse=80000000 00000000 valeur=4E0
```

Notez ici que le bit 64 est mis à 1, ce qui signifie que la mantisse a une valeur de 1.

```
exposant = 0 mantisse=80000000 00000000 valeur=5.0E-1
exposant =-1 mantisse=80000000 00000000 valeur=2.5E-1
exposant =-2 mantisse=80000000 00000000 valeur=1.25E-1
```

Here the values of a half, quarter and an eighth are produced using zero and minus exponents.

3.2.2.5 Contenu de la pile : l'ordre d'affichage

Gobug affiche toujours le contenu des huit registres à virgule flottante, mais l'ordre de l'affichage correspond à la pile elle-même. Ainsi, le registre en haut de la pile est-il toujours appelé "0", et ceci est affiché sur la première ligne montrant le contenu de la pile. Le contenu de ce registre a été le dernier à être placé sur la pile en utilisant une commande "push" en virgule flottante et sera le premier à en être retiré s'il y a une commande "pop" en virgule flottante. Les autres registres sont affichés dans le même ordre que la pile.

3.2.2.6 Mot de contrôle, précision et arrondi

Sur la première ligne de l'écran, vous pouvez voir "Control word: *number* PC=*number (description)* et RC=*number (description)*". Le mot de contrôle est un registre de 16 bits qui contient divers flags, dont les plus importants sont les flags de *contrôle de précision*, les flags de *contrôle d'arrondi* et les masques d'exception. Le programmeur ajuste ces flags lors de l'initialisation de l'unité à virgule flottante en utilisant des instructions à virgule flottante pour s'adapter aux calculs à effectuer et aux types de données utilisés. Les valeurs ont les significations suivantes, qui correspondent aux descriptions que GoBug met entre parenthèses sur la première ligne de l'affichage en virgule flottante :

```
Precision control:
0=24 bits
2=53 bits
3=64 bits
Rounding control:
0=round to nearest or even
1=round down
2=round up
3=chop
```

3.2.2.7 Mot d'état, pointeur de pile et codes de condition

Sur la troisième ligne de l'affichage, vous pouvez voir "Status word: *number* SP=*number* and Condition codes=*number*". Le mot d'état est un registre de 16 bits qui contient des informations sur le résultat de l'exécution de la dernière instruction effectuée par l'unité de calcul en virgule flottante. Les plus importants sont le pointeur de pile, les codes de condition et les flags d'exception. Le pointeur de la pile est essentiellement un compteur indiquant combien des registres ne sont pas utilisés actuellement. Lorsque les registres à virgule flottante sont initialisés (comme ils devraient l'être avant tous les calculs à virgule flottante), le pointeur de la pile est mis à 7. Il décrémente à chaque instruction «push» et s'incrémente à chaque instruction «pop». Vous pouvez ignorer le pointeur de la pile parce que GoBug montre toujours le contenu des registres dans leur ordre correct, voir le paragraphe contenu de la pile: l'ordre d'affichage. Les codes de condition sont au nombre de 4 bits, représentés dans l'ordre C3, C2, C1 et C0. Ils sont utilisés par le processeur pour le branchement conditionnel.

3.2.2.8 Flags et masques d'Exception

Les flags d'exception sont affichés sur la quatrième ligne de l'affichage. Ils proviennent de 7 bits du mot d'état et montrent le résultat des instructions effectuées par l'unité de calcul en virgule flottante. Si ces bits sont à 1, ils apparaissent dans une couleur différente même s'ils n'ont pas changé depuis la dernière instruction en virgule flottante. Notez qu'aucune de ces "exceptions" n'arrête réellement le traitement. Ils ne sont pas non plus mis à zéro à chaque instruction. Ceci permet d'effectuer une série d'instructions avant que les flags soient vérifiés. Cependant, le programmeur doit veiller à effacer les flags au début de l'ensemble d'instructions. Voir, à ce sujet, la remise à zéro des flags d'exception. Chaque flag (autre que le flag de pile) a un bit de "masque" correspondant dans le mot de contrôle. Si vous le souhaitez, le programmeur peut définir les bits de masque pour activer et désactiver les rapports d'exception pour s'adapter aux tests qui peuvent être effectués pendant le traitement. Si le bit de masque correspondant est mis à zéro, le flag d'exception ne sera pas modifié.

Les flags d'exception (dans le mot d'état) et les masques (dans le mot de contrôle) sont :

Bit 5 P – flag de précision (une perte de précision s'est produite)
Bit 4 U – flag d'underflow (résultat trop petit)
Bit 3 O – flag d'overflow (résultat trop important)
Bit 2 Z – flag de division par zéro (division par zéro tentée)
Bit 1 D – flag dénormalisé (opération tentée sur un nombre dénormalisé)
Bit 0 I – indicateur invalide (résultat incertain)

3.2.2.9 Mot de Tag

Le mot de tag est un registre 16 bits dont 2 bits sont réservés pour chaque registre à virgule flottante pour décrire le nombre dans le registre. La signification de ces bits est la suivante:

00 = normal
01 = true
10 = spécial (ce n'est pas un nombre ou denormal)
11 = vide

GoBug affichera "- empty -" si les deux bits sont 11 pour le registre concerné. Dans tous les autres cas, GoBug affichera la valeur du contenu du registre sous forme de nombre réel de la manière habituelle.

3.2.2.10 Nombres réels

Les nombres réels sont des nombres qui peuvent contenir la représentation d'une valeur inférieure à 1. Les nombres à virgule flottante utilisent un point pour ce faire, généralement un point décimal[5]. Dans un nombre à virgule flottante, le point peut être n'importe où parmi les chiffres qui le composent. Cependant, dans le volet de registres en virgule flottante, GoBug donne la valeur du registre en notation scientifique. Sous cette forme, un seul chiffre est autorisé à gauche du point décimal. En notation scientifique, la position correcte du point décimal doit donc être fournie d'une autre manière. Ceci est obtenu en ajoutant un exposant décimal signé à la fin du nombre. Cela correspond à la norme IEEE Floating Point Standard.

Exemples :

1.6789E3 est identique à 1678.9

2.6789E0 est identique à 2.6789

7.6789E-2 est identique à 0.076789

-9.44E7 est identique à -94400000

[5] Bien que la pratique mathématique française impose de convertir le point décimal anglo-saxon en virgule décimale, l'appellation « point décimal » a été conservée ici, s'agissant d'un logiciel anglais dont les affichages demeurent en langue et en format original. On relèvera évidemment une légère contradiction puisqu'il est question *a contrario* de calcul en virgule flottante…

En effet, la représentation correcte d'un nombre réel donné en notation scientifique correspond au *nombre* multiplié par 10 élevé à la puissance de l'exposant.

En théorie, il n'y a pas de limite à la taille de l'exposant, de sorte que des nombres très grands ou très petits peuvent être représentés de cette manière sous forme compacte. En pratique il y a une limite. Voir Modifier les registres FP en utilisant GoBug.

En regardant à nouveau le volet de registres à virgule flottante de GoBug, la colonne de droite montre la valeur correspondante du registre à virgule flottante exprimée en nombre réel selon cette notation scientifique.

3.2.2.11 Précision de conversion de GoBug

Lors du débogage des instructions à virgule flottante, il peut être important d'étudier la précision et l'exactitude du processeur et les instructions qui lui sont données. Par conséquent, lorsque vous affichez le nombre de 80 bits en tant que nombre réel, GoBug tente d'atteindre la précision maximale possible. Certains nombres réels sont plus longs que le volet et doivent être tronqués au niveau de l'affichage. Pour les voir plus précisément élargir le volet en faisant glisser son bord. Pour autant, certains nombres demeurent trop longs pour l'affichage. GoBug affichera un maximum d'environ 80 chiffres, en fonction de la longueur de l'exposant décimal. GoBug prend également cette limite de 80 chiffres comme une précision raisonnable pour les chiffres visibles. L'exactitude réelle de la conversion en nombres réels dans GoBug est la suivante :

- Exposants négatifs ou zéro – 80 chiffres de précision (les calculs sont à 82 octets puis arrondis).
- Les exposants de 1 à 64 – pas de données perdues. La valeur réelle affichée est une représentation précise du nombre de 80 bits.
- Les exposants de 65 et plus – 80 chiffres de précision (les calculs sont à 83/84 chiffres arrondis).

Si vous voulez vraiment voir les chiffres au-delà de la limite visible, vous pouvez cliquer sur le nombre et le visualiser dans la fenêtre de modification comme si vous aviez l'intention de changer les registres FP ou vous pouvez les vider ou les imprimer, auquel cas la valeur ne sera pas tronquée (en fonction du format du papier lors de l'impression !).

3.2.2.12 Modifier les registres FP avec GoBug

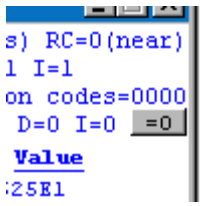
Vous pouvez modifier un registre FP *actif* (ceci est expliqué ci-dessous). Cliquez sur l'exposant, la mantisse ou la valeur, et une fenêtre de modification apparaîtra. Sinon, si le volet FP a le focus sur le clavier, appuyez sur Entrée pour faire apparaître la surbrillance, puis appuyez de nouveau sur Entrée. Vous pouvez utiliser ESC pour en supprimer le contenu, mais si vous souhaitez seulement le modifier, vous pouvez supprimer les chiffres à modifier, saisir les nouveaux chiffres et appuyez sur Entrée pour modifier le contenu du registre. Notez que lorsque vous modifiez la valeur, l'exposant et la mantisse sont susceptibles de changer pour s'adapter à la nouvelle valeur. Cette dernière doit être insérée en tant que nombre réel. Notez que si vous entrez un nombre réel qui a plusieurs chiffres, la valeur qui apparaît dans le volet du registre en virgule flottante peut ne pas correspondre exactement au nombre réel que vous avez entré. Ceci est un comportement normal et se produit parce que le registre à virgule flottante peut ne pas être en mesure de conserver ce nombre particulier avec la même précision que vous l'avez saisi. Le nombre réel qui apparaîtra finalement sera le plus proche du nombre que vous avez entré et que l'unité de calcul en virgule flottante peut atteindre. De plus, si vous entrez un nombre réel avec plus d'un chiffre avant le point décimal, GoBug affichera le nombre réel résultant en format scientifique (un seul chiffre avant le point décimal) avec l'exposant décimal ajusté en conséquence. Entrer de nouvelles valeurs de nombres réels de cette manière et vérifier quel nombre apparaît est une bonne manière d'appréhender de la précision pouvant être obtenue à partir des registres en virgule flottante.

Les maxima et minima qui peuvent être saisis sont :

L'exposant du format virgule flottante : +4001h à –3FFEh
La mantisse : 00000000 00000000 à FFFFFFFF FFFFFFFF.
Les valeurs approximatives en format réel :
3.362103143112093506 3E-4932 à 1.189731495357231765E4932.

Afin de réaliser une modification du registre FP (Floating Point), GoBug doit simuler les instructions du processeur. Pour cette raison, vous pouvez découvrir que cette modification change également le mot d'état, le mot de contrôle et les flags. Des modifications seront normalement apportées à un registre FP *actif*, c'est-à-dire un registre qui stocke déjà une valeur. En revanche, si vous essayez de modifier un registre FP *inactif*, la modification sera toujours faite mais le contenu du registre pourra apparaître comme "vide" dans la mesure où le mot de tag demeurera inchangé. Pour modifier le mot de tag afin que la FPU sache qu'il y a une valeur dans le registre, chargez un nombre dans le registre de la manière habituelle (en utilisant une instruction de chargement) ou insérez une valeur dans le registre par modification du registre MMX correspondant.

3.2.2.13 Mise à zéro des flags d'exception



Le bouton dans le coin supérieur droit de l'écran portant l'indication "= 0" simule l'instruction FCLEX pour mettre tous les flags d'exception à zéro. Vous pouvez utiliser ce bouton pour effacer les flags de sorte que vous pouvez faire un pas sur l'instruction suivante pour voir si elle affecte les flags. Cliquez sur le bouton pour l'activer ou, si le volet FP a le focus, appuyez sur la barre d'espace sur le clavier. Puisque le système ne permet pas de changement direct de ces indicateurs, ceci est réalisé par une instruction FCLEX simulée par le debuggee lui-même.

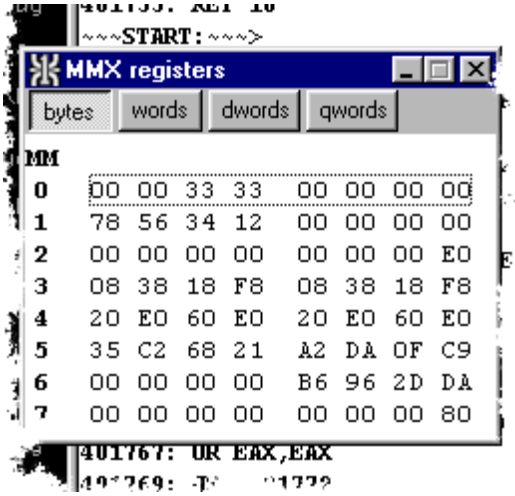
3.2.2.14 L'instruction "WAIT"

Dans votre code désassemblé, il se peut que vous observiez une instruction WAIT (opcode 9Bh) que vous n'avez pas mise dans votre code source. Elle a peut-être été insérée automatiquement par votre assembleur. Cette instruction est utilisée pour indiquer au processeur principal d'attendre que le coprocesseur ait terminé de traiter toutes les exceptions qui se seraient produites lors du traitement de l'instruction en virgule flottante. Normalement, l'instruction WAIT n'est pas nécessaire du tout. Cependant, du temps des co-processeurs arithmétiques (8087, 80287 et 80387), l'instruction WAIT était nécessaire avant chaque instruction en virgule flottante. Cela était dû à une erreur de conception qui a amené le 8086 à envoyer une instruction en virgule flottante au 8087, mais sans attendre que le 8087 ne termine son activité – il continuait simplement à traiter l'instruction suivante qui pourrait être une autre instruction en virgule flottante ou exigeait le les résultats d'un calcul en virgule flottante. La faute avait été corrigée dans le 80287. Je suis reconnaissant à Eric Isaacson pour son explication claire dans son manuel A386. Si votre assembleur insère beaucoup d'instructions WAIT, il pense probablement que vous voulez que le code fonctionne sur un ensemble 8086/8087…

3.2.2.15 Les instructions MMX et les registres

Cela fait un moment que les puces Intel sont en mesure d'exécuter des instructions utilisant les mnémoniques MMX. Celles-ci sont capables de déplacer en une seule fois des blocs de données plus importants que les transferts de 32 bits que les registres ordinaires peuvent effectuer. Les données sont transférées vers et depuis les registres MMX. En fait, ce sont exactement les mêmes registres que ceux utilisés pour les instructions à virgule flottante, sauf que seulement 64 bits sur les 80 disponibles sont utilisés. Cela signifie qu'en pratique, vous ne pouvez pas utiliser les instructions MMX et FPU (unité à virgule flottante) en même temps. Cela signifie également que lorsque vous affichez ces registres en utilisant et les volets à virgule flottante et MMX de GoBug, vous regardez simplement la même chose mais de manière différente.

3.2.2.16 Le volet de registres MMX



Pour afficher le volet de registres MMX, cliquez sur le bouton de la barre d'outils "MMX/FPU", puis choisissez le bouton gauche dans la barre d'outils flottante.

Le volet de registres MMX affiche les registres MMX avec MM0 en haut et MM7 en bas. La manière dont les informations sur les 64 bits sont affichées peut être modifiée. Ici, elles sont affichées sous la forme de huit octets, mais vous pouvez choisir des *mots*, des *dwords* ou un *qword*. Gardez présent à l'esprit que les données apparaîtront à un endroit différent lorsque vous changerez de format en raison de l'effet de stockage inversé.

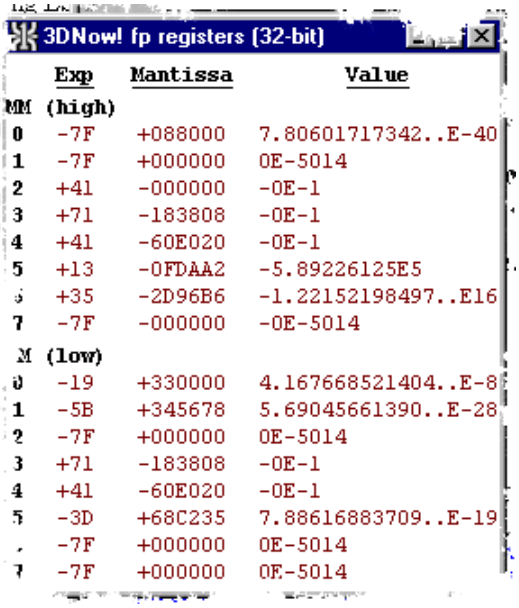
3.2.2.17 Modification des registres MMX avec GoBug

Cliquez sur le registre que vous souhaitez modifier. Cela provoque l'apparition de la fenêtre de modification. Sinon, si le volet MMX a le focus sur le clavier, appuyez sur Entrée pour faire apparaître la surbrillance, puis appuyez à nouveau sur Entrée.

Supprimez les chiffres que vous souhaitez modifier et entrez les nouveaux chiffres, puis appuyez sur "Entrée". Le nombre doit être au format hexadécimal (16 chiffres ou moins). Ne vous inquiétez pas pour les espaces, GoBug les ignore tout simplement. Vous pouvez vous débarrasser de la fenêtre de modification sans la modifier en appuyant sur ESC. Puisque le système ne permet pas à GoBug d'écrire directement dans ces registres, GoBug

doit exécuter une instruction MOVQ pour le compte du debuggee. Pour cette raison lors de la modification, vous pouvez voir apparaître d'autres changements se produire dans les registres MMX. Ces changements ne seront pas significatifs et seront normaux.

3.2.2.18 Le volet de registres 3DNow!



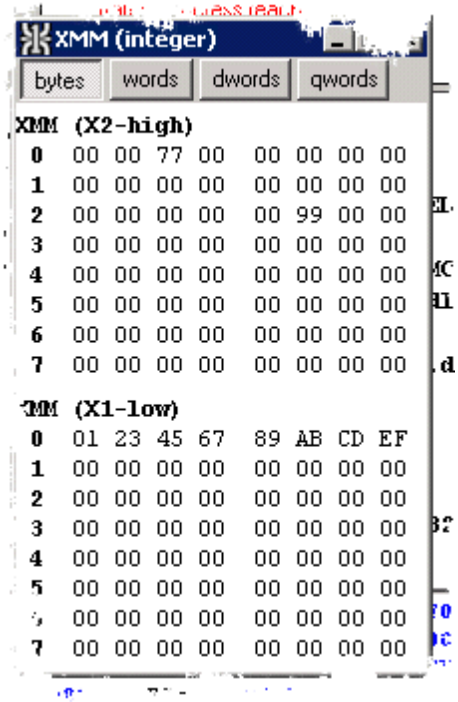
Les registres 3DNow! utilisent les registres FPU mais dans une configuration différente. Actuellement, ils ne sont disponibles que dans les processeurs AMD à partir du K6. Au lieu de huit registres à virgule flottante de 90 bits, ils sont configurés en 16 registres à virgule flottante 32 bits (simple précision). Une différence réelle et utile entre les instructions est que (comme dans les instructions SSE et SSE2) le principe de "pile" par *pushing* et *popping* de données dans les registres est abandonné au profit du placement direct des données, les rendant, de ce fait, beaucoup plus faciles à utiliser.

Pour afficher le volet 3DNow!, cliquez sur le bouton de la barre d'outils "MMX/FPU" puis choisissez le bouton droit dans la barre d'outils flottante.

L'affichage de GoBug montre les registres avec MM0 (haut) en haut et MM7 (bas) en bas. Les instructions 3DNow! n'utilisent pas le mot de contrôle ou les masques et flags d'exception. Aussi, ils ne sont pas affichés.

Vous pouvez modifier le contenu des registres 3DNow! de la même manière que vous modifiez le contenu de la FPU. Si vous n'avez pas de processeur AMD dans votre machine, vous ne pourrez pas le faire.

3.2.2.19 Le volet de registres entiers XMM

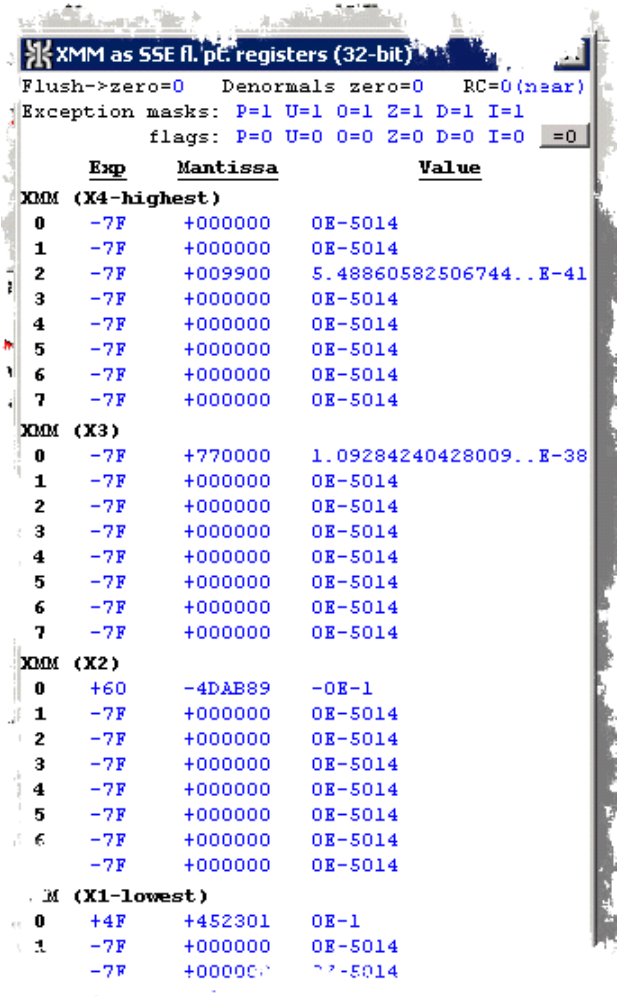


Les instructions d'entiers pour les registres XMM ont été introduites dans le Pentium 4. Lorsqu'ils sont utilisés de cette manière, ils sont considérés comme 16 registres de 64 bits et ce volet en affiche le contenu sous forme d'entiers.

Pour afficher les registres XMM en tant qu'entiers, cliquez sur le bouton de la barre d'outils "XMM registers", puis choisissez le bouton de gauche dans la barre d'outils flottante.

Comme avec les registres MMX vous pouvez choisir le type d'affichage en utilisant les boutons et vous pouvez modifier le contenu des registres lors de l'exécution (en pas-à-pas) en cliquant sur le volet (ou en appuyant sur la touche Entrée lorsque le rectangle de focus apparaît).

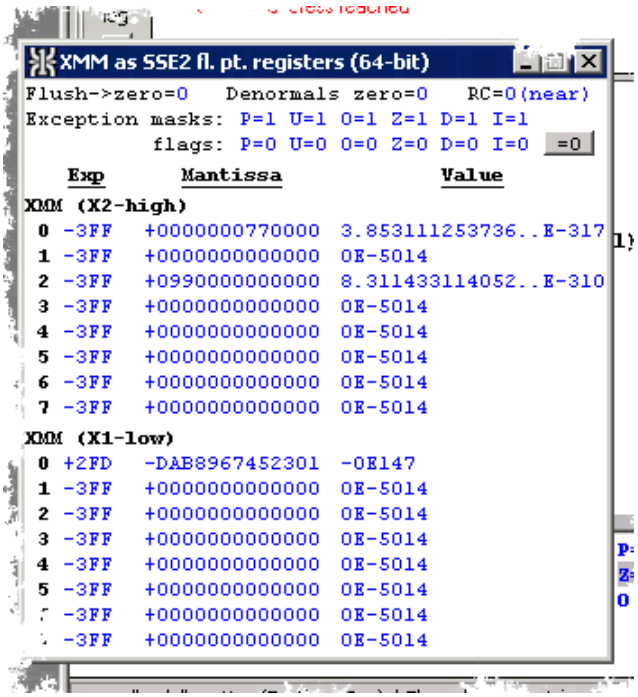
3.2.2.20 Le volet des registres SSE en virgule-flottante



Les instructions SSE utilisent les registres XMM sous la forme de 32 registres 32 bits en virgule flottante (simple précision). Ces instructions ont été introduites dans les processeurs Intel à partir du Pentium III.

Pour afficher le volet de registre à virgule flottante SSE, cliquez sur le bouton de la barre d'outils "XMM registers", puis choisissez le bouton du milieu dans la barre d'outils flottante.

3.2.2.21 Le volet des registres SSE2 en virgule flottante



Les instructions SSE2 utilisent les registres XMM comme 16 registres sz 64 bits chacun à virgule flottante (double précision). Ces instructions ont été introduites dans les processeurs Intel à partir du Pentium 4.

Pour afficher le volet de registre à virgule flottante SSE2, cliquez sur le bouton de la barre d'outils "XMM registers", puis choisissez le bouton droit dans la barre d'outils flottante.

3.2.2.22 Contrôle SSE et SSE2

Il est similaire à celui de l'unité à virgule flottante, mais il existe quelques différences importantes. Les flags de contrôle sont hébergés par le registre MXCSR. Vous pouvez les modifier au moment de l'exécution à l'aide des instructions LDMXCSR (chargement) et STMXCSR (stockage). Le registre MXCSR signale également le résultat des opérations dans les flags d'exception. Comme le FPU, ils sont "collants" et on ne peut compter dessus que s'ils sont effacés immédiatement avant l'instruction qui doit être testée. Lorsque vous effectuez un seul pas, vous pouvez effacer les flags d'exception en cliquant sur le bouton "zero exception flags" (ou en appuyant sur la barre d'espace lorsque le volet a le focus).

Les flags d'exception ont les significations suivantes :

	P	– flag de précision (apparition d'une perte de précision)
	U	– flag d'underflow (résultat trop petit)
	O	– flag d'overflow (résultat trop grand)
	Z	– flag de division par zéro (tentative de division par zéro)
	D	– flag de denormalisation (tentative d'opération sur un nombre dénormalisé)
Bit 0	I	– flag d'état invalide (résultat incertain)

Si le bit de masque correspondant est à 1, aucune exception réelle ne résultera de l'une de ces erreurs (c'est-à-dire que le gestionnaire d'exceptions ne sera pas appelé).

Lorsque le bit de vidage à zéro (flush to zero bit) dans le MXCSR est à 1, le processeur agit différemment dans le cas d'un underflow en virgule flottante. Au lieu de mettre à 1 le flag "O", il retournera un résultat nul dans le registre approprié avec le signe correct, et les flags de précision et d'underflow seront à 1.

Le flag DAZ (denormals are zeroes) est utilisé pour permettre au processeur de passer par-dessus une instruction si un opérande est un nombre virgule-flottante denormal (invalide). Apparemment, cette fonctionnalité est utile pour le traitement en continu des médias. Notez que DAZ était uniquement pris en charge dans les processeurs P4 les plus récents et dans le processeur Xeon. Si vous essayez de le configurer sur un processeur qui ne le supporte pas, le programme va planter. Pour cette raison, il existe un certain code recommandé par Intel pour vérifier la prise en charge de DAZ (voir l'exemple de code Testbug).

Comme dans le FPU, le contrôle d'arrondi peut prendre les valeurs suivantes :

0 = arrondi à la valeur la plus proche (ou paire)
1 = arrondi vers le bas
2 = arrondi
3 = troncage

Il n'y a pas de contrôle de précision dans ces registres (celui-ci est toujours réglé sur 24 bits pour les instructions SSE et 53 bits pour les instructions SSE2).

3.2.2.23 Si les registres ne sont pas disponibles sur votre processeur

Si les volets de registres spéciaux ne sont pas disponibles sur votre processeur, ils apparaîtront toujours mais seront inactifs. Une ligne d'avertissement apparaît en haut du volet pour vous informer à ce sujet.

3.2.2.24 Rafraîchissement des volets de registres spéciaux

Le contenu de tous les volets est actualisé immédiatement après chaque pas. Si vous souhaitez les rafraîchir à un autre moment, utilisez l'élément de menu "inspect, refresh all panes/inspectors" ("inspecter, rafraîchir tous les volets / inspecteurs").

3.2.2.25 Dump des volets de registres spéciaux

Assurez-vous que le volet sur lequel vous souhaitez pratiquer un dump est visible et accédez à l'élément de menu "file, dump". Cliquez sur le volet approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier accueillant les données (GoBug proposera un nom). Choisissez également un chemin d'accès pour que le fichier puisse être écrit (GoBug s'en souviendra la prochaine fois que vous déboguez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI, sauf si vous voulez créer un fichier Unicode). Le fichier créé sur ce ce dump est du texte brut avec des retours chariot, des sauts de ligne, mais pas de tabulations. Lors du dump d'un volet à virgule flottante, la valeur sera affichée aussi précisément que possible dans la précision de conversion de GoBug et ne sera pas tronquée. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur défini sur une police de largeur fixe (par opposition à une police proportionnelle).

3.2.2.26 Impression des volets de registres spéciaux

Assurez-vous que le volet que vous souhaitez imprimer est visible et accédez à l'élément de menu "file, print". Cliquez sur le volet approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Lors de l'impression d'un volet à virgule flottante, la valeur sera

imprimée aussi précisément que possible dans la précision de conversion de GoBug et ne sera pas tronquée. Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.2.3 Le journal de bord et les volets correspondants

3.2.3.1 Que sont les volets de journal de bord ?

Les volets de journal de bord (logpanes) contiennent une trace des événements surnenus dans le debuggee et (si vous le souhaitez) des instructions effectuées par le débogueur.

3.2.3.2 Les 3 niveaux dans le journal principal ; le journal séquentiel

Le journal principal présente les informations du logpane en cascade. Il y a 3 niveaux dans le journal principal :

1. Niveau supérieur : volet de journal de bord montrant les événements principaux et les entrées de l'utilisateur
2. Volets d'événements/messages affichant des événements et des messages
3. Volets d'instructions montrant l'exécution des instructions et les changements de registre qui en résultent.

Le journal séquentiel affiche les événements et les messages dans l'ordre d'apparition. Ceci peut être surveillé en temps réel pour que vous puissiez voir quels messages sont générés par les actions du debuggee.

3.2.3.3 Taille et position du niveau supérieur du journal principal

Pour agrandir temporairement le niveau supérieur du journal principal afin d'afficher son contenu, cliquez sur le bouton bas sous les mots "Log Pane". Autrement, vous pouvez y parvenir de la meme manière en appuyant sur la barre d'espace si la fenêtre principale a le focus. Vous pouvez faire apparaître le journal principal dans une fenêtre séparée en cliquant dessus et en faisant glisser la barre d'armature à gauche des mots "Log Pane". Pour le restaurer dans la fenêtre principale, utilisez l'élément de menu "view". Vous pouvez déplacer le journal principal en bas de la fenêtre principale en utilisant l'élément de menu "view. Pour restaurer le nombre et la position des volets à la valeur par défaut, utilisez l'élément de menu "view, panes as default" ("affichage, volets comme défaut").

3.2.3.4 Comment l'action de débogage affecte ce qui est enregistré (logged)

L'action F5/F6 (un seul pas) et l'action F7 (run/trap/hook/full log) provoquent un enregistrement complet du journal. L'action F8 (run/hook/ part log) provoque la création d'un journal des événements et des messages, mais pas des instructions ni de changements de registre. L'action F9 (exécutée en arrière-plan) trace uniquement les exceptions.

3.2.3.5 Liste des choses traçables (loggable things)

Voici une liste de choses qui apparaissent dans le journal principal avec des explications si nécessaire :

Volet de journal de bord de niveau supérieur

"Running to starting address" ("Fonctionnement jusqu'à l'adresse de départ")
c'est la première entrée affichée lors du chargement du debuggee

"Starting address reached" ("Adresse de départ atteinte")
ceci apparaît une fois que le debuggee a atteint son adresse de départ

User's key input Entrée de clavier de l'utilisateur
montrant le nom de la touche et l'état de la touché Shift

Mouse input Entrée de la souris
bouton enfoncé et mouvement de la molette

Point d'arrêt atteint

indiquant le type de point d'arrêt, l'adresse ou la procédure de code, ou le nom du message

Hot-key pressed Touche de raccourci actionnée

Chaînes envoyées au débogueur à l'aide de l'API *OutputDebugString*

Les exceptions

Les erreurs d'API

Volet de journal de bord d'Événement/message

Messages à n'importe quelle fenêtre

Fenêtres sur le point d'être créées

Fenêtres sur le point d'être activées

Fenêtres sur le point d'être déplacées ou dimensionnées

Fenêtres sur le point d'être minimisées ou maximisées

Focus de fenêtre sur le point de changer

Fenêtres sur le point d'être détruites

Une DLL (ou un EXE) en cours de chargement dans l'espace adressage du debuggee

Une DLL (ou un EXE) en train d'être déchargé dans l'espace adressage du debuggee

Démarrage d'un nouveau thread

Achèvement d'un thread

Un thread devenant inactif (plus aucun message en attente d'être traité par le thread)

Chaînes envoyées au débogueur à l'aide de l'API *OutputDebugString*

Les exceptions

Les erreurs d'API

Volet de journal de bord d'Instructions

Instructions qui ont été exécutées

Message d'appel du thread et réception du thread (si un message a précédé les instructions)

Un retour d'une instruction CALL

"Maintenant à ..." (s'il y a eu une interruption dans la séquence d'instructions, par exemple un F6 saute par-dessus un pas)

Démarrage d'un nouveau thread

Achèvement d'un thread

Chaînes envoyées au débogueur à l'aide de l'API *OutputDebugString*

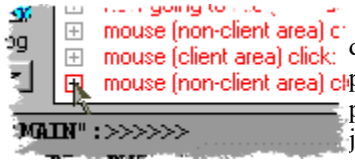
Les exceptions

Les erreurs d'API

Volet de journal de bord séquentiel

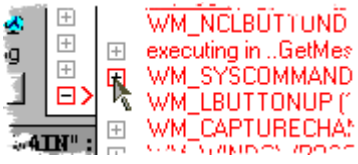
Toutes les entrées affichées dans le volet de journal de bord de niveau supérieur et dans le journal des événements/messages, mais en séquence.

3.2.3.6 Affichage d'un journal des événements/messages



Déplacez le curseur sur un bouton "plus" sur le volet de journal principal. Il deviendra rouge. Cliquez dessus pendant qu'il est rouge. Cela provoquera l'apparition du journal des événements/messages associé à l'entrée dans le journal principal. S'il n'y a pas de bouton "plus" à côté d'une entrée dans le volet de journal principal, c'est parce que le journal de l'événement/message est vide pour cette entrée. Cela peut être dû au fait qu'aucun événement ou message ne s'est produit après l'entrée, ou parce qu'il y a peu d'informations enregistrées car vous utilisez l'action F9 (exécution en arrière-plan) ou parce que vous exécutez un point d'arrêt et que point de break n'a pas encore été atteint. Pour modifier le journal des événements/messages que vous souhaitez afficher, cliquez sur un autre bouton "plus". Pour supprimer le jounal de l'événement/ message, cliquez à nouveau sur le bouton qui affichera maintenant un "moins"

3.2.3.7 Affichage d'un journal des instructions



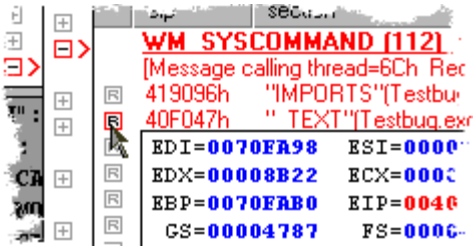
Déplacez le curseur sur un bouton "plus" dans le journal des événements/messages qui s'affiche. Il deviendra rouge. Cliquez dessus pendant qu'il

est rouge. Cela provoquera l'apparition du volet de journal d'instructions relatif à l'entrée dans le journal des événements/messages. S'il n'y a pas de bouton "plus" à côté d'une entrée dans le journal des événements/messages, c'est parce que le journal des instructions est vide pour cette entrée. Il peut y avoir plusieurs raisons à cela. Il se peut que vous ayez appuyé sur F8 (run/ hook/ part log) qui ne consigne pas les instructions. Ou il se peut que vous utilisiez F5/F6 (en une seule étape) ou F7 (journal complet) mais le message est allé à une procédure de fenêtre en mémoire partagée : si c'est le cas, les instructions ne seraient pas consignées. Il se peut enfin que vous utilisiez un break de message pas-à-pas, mais vous avez appuyé sur F6 pour ignorer le message concerné. Pour changer le journal d'instruction que vous souhaitez voir, cliquez sur un autre bouton "plus". Pour supprimer le journal de l'événement/message, cliquez à nouveau sur le bouton qui affichera maintenant un "moins".

3.2.3.8 Déplacement des volets de journaux d'événements / messages / instructions

La seule façon de le faire est de déplacer la fenêtre parent. Dans le cas d'un volet de journal intégré, déplacez la fenêtre principale de GoBug. Dans le cas d'un volet de journal séparé, déplacez-le.

3.2.3.9 Visualisation des modifications dans les valeurs de registre



Déplacez le curseur sur un bouton "R" sur le volet de journal d'instruction qui s'affiche. Les registres et les flags apparaîtront avec tous les changements affichés dans une couleur différente. Si l'instruction dans le journal est une instruction MMX ou en virgule flottante (c'est-à-dire utilisant la FPU), les registres appropriés apparaîtront. Dans sa version actuelle, le journal n'affiche pas le contenu des registres XMM.

3.2.3.10 Utilisation du clavier dans le journal

Au lieu d'utiliser la souris pour effectuer des actions dans le journal, vous pouvez utiliser les touches ou combinaisons de touches du clavier suivantes :

TAB – si le volet de journal fait partie intégrante de la fenêtre principale, déplace le focus sur le volet de journal principal

CrtlTAB – si le volet de journal est séparé, déplace le focus sur le volet de journal à partir d'autres volets et fenêtres

TAB – déplace le focus en avant entre les volet de journal ouverts

ShftTAB – déplace le focus en arrière entre les volet de journal ouverts

UpArrow et **DnArrow** – déplace le rectangle de focus vers le haut et vers le bas parmi les éléments du volet de journal qui a actuellement le focus

CrtlUpArrow et **CrtlDnArrow** – déplace le rectangle de focus sur la première ou la dernière ligne

RtArrow – ouvre le volet de journal si le bouton "plus" est rouge, ou affiche les valeurs du registre si le bouton "R" est rouge

LeftArrow – ferme le volet de journal en cours de visualisation ou ferme le volet de valeur du registre s'il est affiché

PgUp et **PgDn** – fait défiler vers le haut ou vers le bas du volet de journal s'il y a une barre de défilement

CrtlPgUp et **CrtlPgDn** – fait défiler rapidement le volet de journal s'il y a une barre de défilement

3.2.3.11 Ignorer certains messages

En mode F7, le journal peut se voir rempli par le mouvement de la souris, HITTEST, SETCURSOR et des messages similaires qui ne présentent généralement que peu d'intérêt. Ces messages sont filtrés par défaut mais vous pouvez modifier la liste des messages ignorés en cliquant sur l'élément de menu "settings, log settings" ("paramètres, paramètres de journal") et l'onglet "ignores". Notez que le message POWERBROADCAST est uniquement envoyé aux machines alimentées par batterie. Tous les messages ignorés par le journal seront également ignorés par le break de message en un seul pas.

3.2.3.12 Couleurs du texte du journal et police de caractère

Les entrées de journal relatives au thread principal apparaissent dans la couleur du texte du journal principal. Les entrées de journal relatives à d'autres threads ont une couleur qui est liée à la *séquence dans laquelle le thread a été créé*. Vous pouvez voir cette séquence dans la feuille de propriétés qui apparaît lorsque vous cliquez sur l'élément de menu "settings, log settings" ("paramètres, paramètres de journal") et l'onglet "text". Le point de départ de la séquence de couleurs change en fonction de la couleur du texte principal. Vous pouvez changer la séquence en cliquant sur les boutons "shift". Vous pouvez également modifier la couleur et la police du texte principal à l'aide de cette feuille de propriétés.

3.2.3.13 Suspension du journal et du débogage pendant la visualisation

L'ouverture des sous-volets de journal arrête tout autre enregistrement de journal en cours. L'icône du journal se transforme alors en un flocon de neige pour l'indiquer. Si le debuggee est en cours d'exécution en arrière-plan, il ne sera pas affecté car il n'y a normalement aucune activité de journal de toute façon. Mais si vous consignez des messages (action F8) ou des instructions (action F7), la suspension du journal interrompt le debuggee. Le journal sera également suspendu pendant un vidage ou une impression du contenu du journal. De même, lorsque vous consultez le journal séquentiel, vous pouvez figer le journal en cochant la case dans le volet de jounal séquentiel.

Sauf si vous faites un pas simple, si une fenêtre du debuggee *se trouve derrière* le sous-volet de journal sur l'écran, la suppression de la fenêtre entraînera une activité du debuggee. Cela est dû au fait que le système va provoquer la peinture des parties des fenêtres du debuggee qui sont ensuite découvertes par la suppression du sous-volet de journal. Cela va, bien sûr, changer le contenu du journal si vous utilisez l'action F7 ou F8. En pratique, cela ne devrait pas poser de problème mais vous pouvez éviter cela en minimisant d'abord le debuggee, avant de regarder le sous-volet de journal.

3.2.3.14 Mémoire utilisée par le journal

Pour modifier la quantité de mémoire utilisée par le journal, cliquez sur l'élément de menu "Settings, log settings" ("Paramètres, paramètres du journal"), puis sur l'onglet "Memory". Vous pouvez faire en sorte que GoBug puisse choisir la quantité de mémoire à utiliser. Dans ce cas, il est fort probable que GoBug prendra un maximum de 640K (après vérification de la capacité mémoire de la machine). Ce n'est pas pris d'un seul coup mais par étapes successive de 4K. Dans la plupart des sessions de débogage courtes, seule une partie de cette mémoire sera réellement utilisée. 640 Ko de mémoire permettront d'enregistrer environ 30 720 lignes d'événements, de messages et d'instructions avec les changements de registre. Cette mémoire sera utilisée plus rapidement si vous utilisez F7 qui enregistre toutes les instructions et les modifications. L'utilisation de points d'arrêt (breakpoints) pour accéder rapidement au code qui vous intéresse préserve la compacité du journal, car rien n'est consigné lors de l'exécution d'un point d'arrêt. Si vous souhaitez limiter la quantité de mémoire utilisée par le journal, vous pouvez le définir manuellement.

3.2.3.15 Mise en boucle des enregistrements lorsque la mémoire est utilisée

Vous pouvez voir ce qui suit dans le journal : "part of the record looped away" ("partie de l'enregistrement en boucle"). Cela signifie que la mémoire du journal est pleine et que, pour faire de la place pour les enregistrements ultérieurs, les enregistrements antérieurs ont dû être abandonnés. Ce n'est généralement pas un problème, mais vous pouvez réduire la quantité de mémoire de journal utilisée en cours d'exécution à la zone de code qui vous intéresse plus particulièrement à l'aide de points d'arrêt (breakpoints). Cela est dû au fait que rien n'est consigné lors de l'exécution d'un point d'arrêt.

3.2.3.16 Vidage du journal

Vous pouvez effacer l'intégralité de la mémoire utilisée par le journal en utilisant l'élément de menu "settings, clear log" ("paramètres, effacer le journal"). Cela restaure également au système toute mémoire utilisée par le journal.

3.2.3.17 "Execution ..." et "Executing in ...'START' .."

Si des instructions n'ont pas été exécutées en réponse à un message, les mots "Executing in ..." peuvent apparaître dans le journal principal ou dans le journal des événements/messages. Un exemple serait celui d'un seul pas effectué après le démarrage lorsque le debuggee n'a pas encore atteint la boucle de message, lorsque "Executing in ..'START' .." apparaîtra. Cette formule peut également apparaître si une fonction provoque des messages et que les instructions reviennent à la fonction après que ces messages ont été traités. Dans les applications de console (qui n'ont pas de messages), vous verrez cette entrée beaucoup plus fréquemment.

3.2.3.18 GetMessage, DispatchMessage et mise en file d'attente de messages

La mise en file d'attente des messages retournés par *GetMessage* n'indique pas nécessairement qu'ils seront envoyés à la procédure de fenêtre par la boucle de message. Notamment, ils peuvent être modifiés avant d'être envoyés à celle-ci (par exemple par *TranslateMessage*). Les messages mis en file d'attente dans la boucle de message sont consignés dans le journal, et s'ils arrivent à la procédure de fenêtre, ils sont de nouveau consignés. La séquence d'événements affichée par le journal pour les messages en file d'attente est la suivante :

1. Message en file d'attente consigné lorsque GetMessage retourne
2. Si F5/6 (pas-à-pas) ou F7 (journal complet) les instructions exécutées dans la boucle de message sont consignées, et enfin
3. Si le message arrive à la procédure de la fenêtre, elles sont de nouveau enregistrées (la source est montrée dans le journal comme "en file d'attente").

3.2.3.19 Message ne montrant aucune instruction

Généralement, si vous avez utilisé l'action F8 (Run / hook / log part), tous les messages autres que ceux qui sont ignorés apparaîtront dans le journal des événements/messages, mais aucune instruction ne sera consignée. De même, si vous avez effectué un pas unique avec le break de message en une seule étape à l'état "on" et que vous avez appuyé sur F6 (ignore), aucune instruction ne sera enregistrée non plus. Toutefois, si vous appuyez sur F5 lors d'un break de message en une seule étape, les instructions seront consignées. F7 enregistrera toutes les instructions dans la procédure de fenêtre autres que celles exécutées dans la mémoire partagée ou dans le code d'une API.

3.2.3.20 Résolution du cas de messages de même valeur

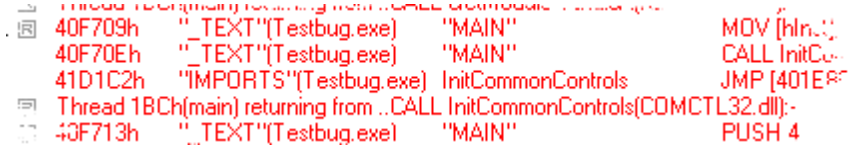
Un certain nombre de messages dans Windows ont la même valeur. Cela n'a pas d'importance car ils sont identifiables comme destinés à ou provenant de fenêtres d'un certain type. Par exemple, les messages TB_ENABLEBUTTON et TTM_ACTIVATE ont tous deux la valeur 401h. En fait, il y a 11 messages avec cette valeur. Avec de tels messages, GoBug découvre quel type de fenêtre est impliqué et enregistre le nom du message correct en conséquence.

3.2.3.21 La consignation de la commande dans un journal et la notifiecation des messages

Quatre messages, WM_COMMAND, WM_NOTIFY, WM_DEVICECHANGE et WM_POWERBROADCAST, sont des transporteurs pour d'autres messages. Le journal affiche d'abord l'autre message et conclut en indiquant le transporteur entre parenthèses.

3.2.3.22 La consignation dans un journal des appels d'API

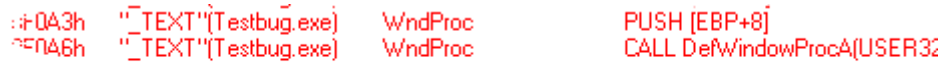
Attendez-vous à voir des instructions supplémentaires dans le journal qui n'apparaissent pas dans le volet de code autour des appels d'API. En effet, un appel à une fonction importée n'est pas effectué directement, l'exécution étant routée vers une autre partie du fichier exécutable où la véritable adresse de la fonction importée a été placée par le système lors du chargement du programme. La méthode précise de réacheminement utilisée et la position de l'adresse réelle de la fonction importée varie d'un éditeur de liens à l'autre, mais un appel API peut généralement apparaître si vous utilisez l'action F7 (run / hook / trap / log full) :



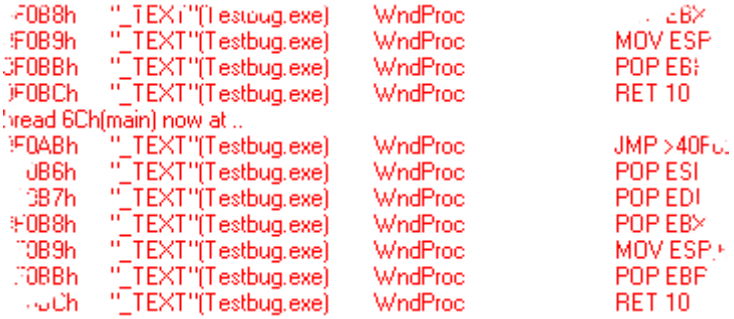
Dans la mesure où GoBug fonctionne d'une manière légèrement différente en sautant par-dessus l'appel en utilisant l'action F6, l'instruction intermédiaire juste avant l'appel n'apparaîtra pas.

3.2.3.23 Récursion dans la procédure de fenêtre

Vous pouvez voir que les instructions pour un message particulier se terminent soudainement pendant un message particulier comme dans cet exemple qui s'est produit pendant WM_NCACTIVATE :



Voici ce qui s'est passé : le système a envoyé un autre message depuis l'API *DefWindowProc*. En fait, le message est WM_GETTEXT, de sorte que le système peut dessiner le titre de la fenêtre. Si nous regardons maintenant le journal d'instructions pour WM_GETTEXT, nous pouvons voir que la dernière instruction pour WM_GETTEXT est, en fait, le RET issu de la procédure de fenêtre, mais alors l'exécution semble sortir de la procédure de fenêtre une deuxième fois. C'est parce que l'appel original à *DefWindowProc* (pendant le message NCACTIVATE) est maintenant retourné :



3.2.3.24 *Le journal séquentiel*

Ceci est une représentation séparée de certaines parties de l'information du journal. Le volet de journal séquentiel affiche les événements et les messages dans un seul volet et dans l'ordre d'apparition. Vous pouvez créer le journal séquentiel en utilisant l'élément de menu "view, sequential log". Dans ce journal, vous pouvez voir en temps réel quels événements et messages sont causés par les actions du debuggee. Pour vous aider à les observer, il est possible de faire apparaître le journal séquentiel toujours au-dessus du debuggee, en utilisant l'élément de menu "view, log (tout en haut)", ou la case à cocher dans le volet de journal séquentiel lui-même. Il est également plus facile de suivre l'ordre réel des messages du journal séquentiel, ce qui est souvent important. Moins d'informations apparaissent dans le journal séquentiel que dans le journal principal, mais vous pouvez toujours utiliser le journal principal pour obtenir toutes les informations dont vous avez besoin.

Pour éviter la mise à jour du journal séquentiel lors de la visualisation, cochez la case "freeze log" ("gel du journal"). L'icône du journal se transforme en un flocon de neige pour attester de cette action. N'oubliez pas de décocher la case lorsque vous voulez que le journal continue.

3.2.3.25 *Vidage du volet de journal*

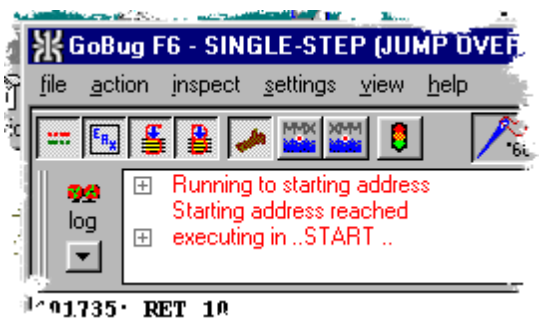
Assurez-vous que le volet de journal que vous souhaitez vider est visible et accédez à l'élément de menu "file, dump" ("fichier, vidage"). Cliquez sur le volet de journal approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "vider le volet". Si le journal contient également des valeurs de registre, celles-ci peuvent également être sauvegardées. Choisissez un nom de fichier pour le fichier de vidage (GoBug proposera un nom). Choisissez un chemin d'accès pour que le fichier soit enregistré (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains). Changez le point de départ et la taille du vidage. Le fichier créé sur ce vidage est du texte brut avec des retours chariot et des sauts de ligne mais pas de tabulations. Il est généralement préférable d'afficher le fichier de vidage avec un éditeur parameter avec une police de largeur fixe (par opposition à une police proportionnelle).

3.2.3.26 *Impression du volet de journal*

Assurez-vous que le volet de journal que vous souhaitez imprimer est visible et accédez à l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur le volet de journal approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Si le journal contient des valeurs de registre, celles-ci peuvent également être imprimées. Changez le point de départ et la taille de l'impression. Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.2.4 La barre d'outils et ses boutons (Toolbar)

3.2.4.1 *Les boutons de la barre d'outils*



Dans la figure ci-contre, les boutons de la barre d'outils commandent respectivement, de gauche à droite : le volet de code, le volet de registre, le volet de pile ESP, le volet de pile EBP, le volet de journal, le volet de registre MMX/FPU (également 3DNow!), les volets de registre XMM (XMM, SSE et SSE2), les feux de circulation et le thread.

3.2.4.2 Le rôle des boutons de commande de volet

Ces boutons doivent être utilisés pour faire apparaître ou disparaître les different volets. Vous pouvez changer la position des volets dans la fenêtre principale en les supprimant tous, puis en les réinstallant dans l'ordre qui vous convient. Ils réapparaissent dans l'ordre suivant : en haut à gauche, en bas à gauche, en haut à droite, en bas à droite.

3.2.4.3 Le bouton de feux tricolores

Utilisez ce bouton pour afficher et contrôler l'action en cours des threads. Pour plus d'informations, voir le contrôle des feux de circulation.

3.2.4.4 Le (ou les) boutons(s) de thread

Chaque thread dispose d'un bouton. Utilisez ces boutons pour examiner l'état actuel d'un thread aussi loin que GoBug le rend possible. Pour plus d'informations, voir les boutons de thread

3.2.5 La touche de raccourci (hot-key)

La touche de raccourci par défaut de GoBug est Shift+Pause. Sur certains claviers, cela peut être affiché comme Shift+Break. Vous pouvez voir quelle touche de raccourci est actuellement définie en cliquant sur l'élément de menu "settings, define hot-key" ("paramètres, définir une touche de raccourci").

3.2.5.1 Fonction de la touche de raccourci

La touche de raccourci exécute les actions suivantes :

- Elle met GoBug au premier plan. Bien que GoBug fasse tout son possible pour se positionner au premier plan, le système essaye en fait de se prémunir contre toute application qui se mettrait au premier plan et parfois cela ne sera pas autorisé. Si cela se produit, vous devrez cliquer sur la barre des tâches pour placer GoBug au premier plan.
- Elle tente d'obtenir le debuggee dans la porte d'entrée s'il n'est pas déjà là. Ceci est réalisé en envoyant au debuggee un message enregistré qui n'a d'autre effet que de s'assurer que *GetMessage* retourne dans la boucle de message. Si cela ne fonctionne pas, ou si le flag de trap a été libéré, GoBug inondera les zones de code non partagées d'INT 3 dans le but de capter un thread dans la boucle de débogage. De cette façon, un thread qui se trouverait coincé dans une boucle continue dans son propre code (par opposition à une DLL système) sera capturé par la touche de raccourci.

3.2.5.2 Comment modifier la touche de raccourci

Cliquez sur l'élément de menu "settings, define hot-key" ("paramètres, définir une touche de raccourci"). Appuyez sur les touches constituant la nouvelle touche de raccourci. L'écran vous indiquera si vous pouvez ou non utiliser cette touche (ou combinaison de touches) de raccourci. Si vous voulez utiliser cette touche, cliquez sur "apply". Il est préférable d'utiliser une combinaison inhabituelle de touches afin que la touche de raccourci ne puisse pas être invoquée en utilisation normale. Crtl et/ou Shift et une touche de fonction F donnent généralement une combinaison satisfaisante. Évitez d'utiliser des combinaisons utilisées par Windows par exemple. Alt-Tab, Alt-Shft-Tab, Alt-F4 ou Alt-Esc ou des combinaisons de touches utilisées comme raccourcis par l'application que vous avez l'intention de déboguer.

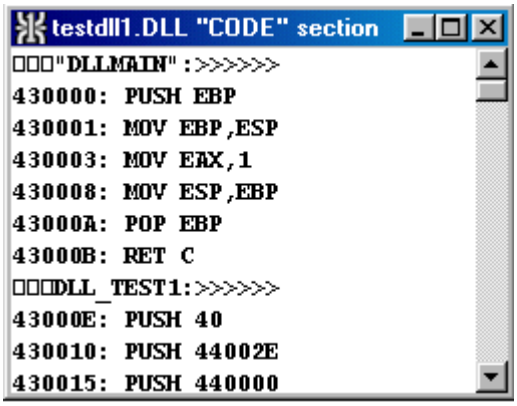
3.2.5.3 Interface clavier de la boîte de dialogue et touches raccourci clavier

Tab, Esc, Enter et les touches de flèches sont réservées pour vous permettre de naviguer et de sélectionner dans la boîte de dialogue de la manière habituelle, mais les combinaisons de ces touches avec Shift, Crtl ou Alt peuvent être utilisées comme touches de raccourci.

3.2.6 Inspecteurs de code, données et symboles

3.2.6.1 Inspecteur de code

L'*inspecteur de code* est utilisé pour afficher et désassembler le code 32 bits chargé dans l'espace d'adressage du debuggee ou dans la mémoire partagée :

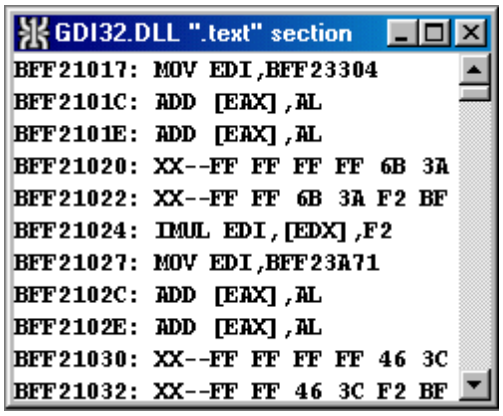


3.2.6.1.1 Création d'un inspecteur de code

Il y a 5 manières possibles de créer un inspecteur de code :

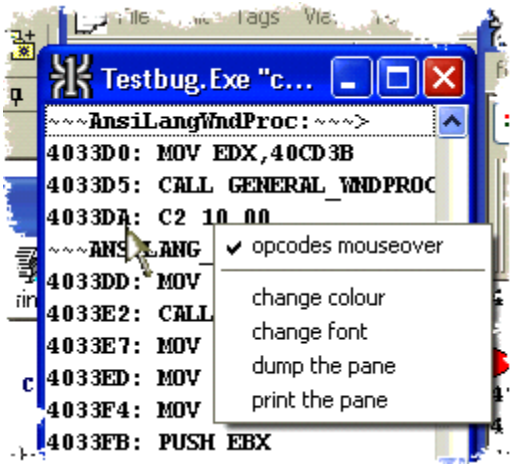
- En cliquant sur le bouton "Afficher en tant que code" dans la boîte de dialogue du symbole de code, après avoir sélectionné un symbole de code dans la boîte de dialogue. Cela ne fonctionne que si les symboles sont chargés.
- En cliquant sur l'élément de menu "inspect, code by address" ("inspecter, code par adresse"). Cela crée une boîte de dialogue qui affiche les sections de code et permet d'entrer une adresse pour l'inspecteur de code.
- En cliquant sur le bouton "View as code" ("Afficher en tant que code") dans la boîte de dialogue des adresses debuggee et DLL, après avoir sélectionné une zone appropriée de L'EXE DLL à inspecter.
- Par le bouton "View as code" ("Afficher en tant que code") dans la boîte de dialogue de recherche/promenade, après avoir sélectionné une adresse dans la liste des résultats. Vous devez soit avoir cliqué sur "promenade" ou effectué une recherche d'abord !
- En cliquant avec le bouton droit de la souris sur le volet de registre ou sur un volet de pile de sorte qu'un menu contextuel apparaisse, puis en cliquant sur l'élément de menu "inspect code at ....." ("inspecter le code au .....").

3.2.6.1.2 Code non résolu dans un inspecteur de code



Si l'inspecteur de code ne peut pas convertir le code en mnémoniques, la raison la plus probable est que le code est, en fait, du *code 16 bits*, plutôt que du *code 32 bits*. Dans ce cas, il peut produire le type d'affichage ci-contre.

Ici, on voit que GoBug affiche des lignes qu'il ne peut pas résoudre. Celles-ci commencent par "XX--" suivi par les 8 opcodes suivants (seulement 6 visibles sur ce volet).



3.2.6.1.3 Dernier code visible dans un inspecteur de code

La dernière zone de code visible dans un inspecteur de code marque la fin des données brutes dans la section. Après cela, la section peut être complétée avec des zéros jusqu'à la fin de la section en mémoire : ces zones ne sont pas affichées par l'inspecteur de code. Certains programmes peuvent utiliser cette zone de mémoire pour stocker des données. Pour visualiser cette partie de la section, utilisez un inspecteur de données.

3.2.6.1.4 Affichage des opcodes dans un inspecteur de code

Vous pouvez afficher les opcodes des lignes individuelles dans un inspecteur de code en activant le commutateur "opcodes mouseover". Cela provoque l'affichage des opcodes sous le cur-

seur de la souris. Vous pouvez trouver ce commutateur dans le menu contextuel si vous faites un clic droit sur l'inspecteur.

3.2.6.2 Inspecteur de données

L'*inspecteur de données* peut être utilisé pour visualiser toute zone de mémoire lisible dans l'espace d'adressage du debuggee ou dans la mémoire partagée. La vue est octet par octet. En raison du principe de stockage inverse, cela signifie que dwords et mots apparaîtront à l'envers, c'est à dire qu'un mot de valeur 12345678h apparaîtra comme 78 56 34 12, et un mot de valeur 1234h apparaîtra comme 34 12. Les octets apparaîtront dans le bon ordre.

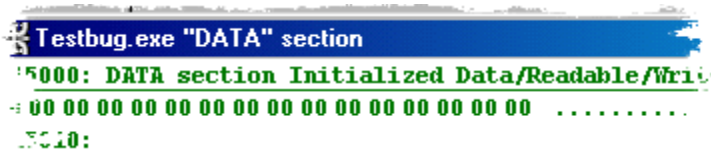
3.2.6.2.1 Création d'un inspecteur de données

Il existe plusieurs façons de créer un inspecteur de données :

- En cliquant sur le bouton "View as data" ("Afficher en tant que données") dans la boîte de dialogue de symbole de données, après avoir sélectionné un symbole de données dans la boîte de dialogue. Cela ne fonctionne que si les symboles sont chargés.
- En cliquant sur l'élément de menu "inspect, data by address" ("inspecter, données par adresse"). Cela crée une boîte de dialogue qui vous permet d'entrer une adresse 32 bits pour l'inspecteur de données.
- En cliquant sur l'élément de menu "inspect, data by selector/handle" ("inspecter, données par sélecteur/handle"). Cela crée une boîte de dialogue qui vous permet d'entrer un sélecteur ou un handle 16 bits. GoBug appelle l'API *GetThreadSelectorEntry* pour résoudre cette valeur de 16 bits. Notez que, dans les versions ultérieures de Windows, les handles de sélecteur ne sont généralement pas utilisés.
- En cliquant sur le bouton "View as data" ("Afficher en tant que données") dans la boîte de dialogue des adresses debuggee et DLL, après avoir sélectionné une zone appropriée de l'EXE/DLL à inspecter.
- En cliquant sur le bouton "View as data" ("Afficher en tant que données") dans la boîte de dialogue de recherche/promenade, après avoir sélectionné une adresse dans la liste des résultats. Vous devez d'abord soit avoir cliqué sur "promenade", soit avoir effectué une recherche !
- Si l'adresse que vous souhaitez voir est actuellement dans un registre, vous pouvez en utiliser le contenu pour insérer l'adresse directement, en utilisant l'élément de menu "inspect, data by register" ("inspecter, données par registre" dans le menu principal.
- En cliquant avec le bouton droit de la souris sur le volet de registre ou sur un volet de pile pour qu'un menu contextuel apparaisse, puis en cliquant sur l'élément de menu "inspect data at ....." ("inspecter les données à .....").
- Vous pouvez également créer un type spécial d'inspecteur de données en affichant une ressource à partir de la ressource de dialogue.

3.2.6.2.2 Types spéciaux d'inspecteurs de données

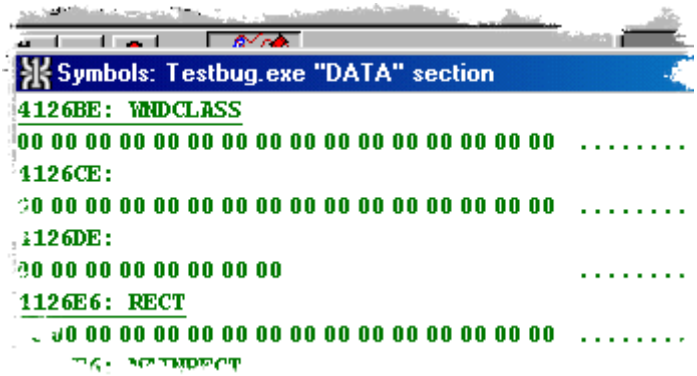
L'inspecteur de données peut montrer ce que l'on sait de la zone actuellement examinée. Par exemple :



Ici, GoBug sait que l'adresse regardée se trouve dans la section "DATA" du debuggee.

Si l'inspecteur de données a été créé à l'aide de l'élément de menu "inspect, data by register" ("inspecteur, données par registre", vous pouvez suivre le registre en cliquant sur le bouton de suivi. Ceci renouvellera le contenu de l'inspecteur à partir de la valeur courante du registre concerné.

Si l'inspecteur de données a été créé à l'aide de la boîte de dialogue du symbole de données, l'inspecteur produit un affichage différent – en commençant une nouvelle ligne pour chaque symbole comme, par exemple :



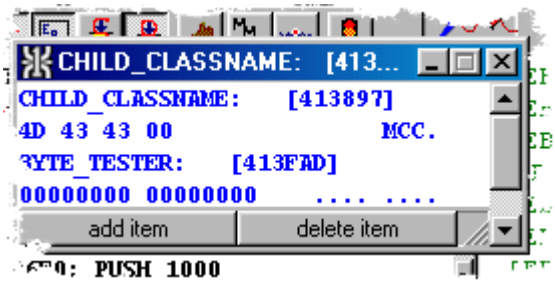
Notez que ce type d'inspecteur affiche les octets dans la zone de données dans l'ordre. Si vous voulez voir des symboles qui ne sont pas en séquence, utilisez un inspecteur de symboles (voir ci-dessous).

3.2.6.3 Inspecteur de symbole

L'*inspecteur de symboles* est utilisé pour afficher le contenu des symboles de données lors du débogage. Avec ce type d'inspecteur, vous pouvez voir plusieurs symboles dans un inspecteur même si leurs adresses ne sont pas adjacentes.

3.2.6.10 Création d'un inspecteur de symbole

Utilisez l'élément de menu "inspect, new symbol inspector" ("inspecter, nouvel inspecteur de symbole"). Dans l'inspecteur de symboles, cliquez sur "add item" ("ajouter un élément") pour ajouter un symbole à la liste des symboles dans l'inspecteur. Il vous sera demandé le format selon lequel vous voulez que le symbole soit représenté (octets, mots ou dwords). Pour supprimer un symbole, cliquez dessus pour le mettre en surbrillance, puis cliquez sur "delete item" ("supprimer l'élément").



Dans l'inspecteur de symbole représenté ici, le premier symbole est "CHILD_CLASSNAME", qui est à l'adresse 413897h. Il est représenté selon le format BYTE et consiste en une chaîne de 4 octets contenant "MCC" terminée par un caractère nul. BYTE_TESTER est une structure de 2 mots à l'adresse 413FADh. Notez que l'inspecteur accueille le dernier symbole ("CHILD_CLASSNAME") en son titre afin de pouvoir l'identifier facilement.

3.2.6.11 Taille affichée de chaque symbole

GoBug évalue la taille de chaque symbole à partir des informations de débogage incorporées dans le debuggee, ou à partir du fichier de symboles (fichier Dbg ou Map). S'il y a deux symboles à la même adresse de données, GoBug considérera le premier comme un simple label ne contenant aucune donnée. Utilisez le second nom de symbole dans l'inspecteur de symboles.

Si GoBug a pris les symboles d'un fichier map, il se peut que GoBug ne connaisse pas la taille du tout dernier symbole dans le fichier map et qu'il puisse être mal représenté dans l'inspecteur de symboles.

3.2.6.12 Utilisation du clavier dans l'inspecteur de symbole

Au lieu d'utiliser la souris pour effectuer des actions dans l'inspecteur de symboles, vous pouvez utiliser les touches ou combinaisons de touches suivantes :

a – ajouter un élément

d – supprimer un élément en surbrillance

TAB – mettre en surbrillance un élément visible

ShftTAB – supprimer la surbrillance

Voir aussi défilement des inspecteurs.

3.2.6.4 Limitation du nombre d'inspecteurs

En théorie, il n'y a aucune limitation de cet ordre.

3.2.6.5 Défilement des inspecteurs (scrolling)

À l'aide de la souris, vous pouvez entreprendre un défilement en cliquant sur les flèches de la barre de défilement ou sur la barre de défilement ou en faisant glisser la boîte de défilement.

L'interface du clavier (lorsque l'inspecteur a le focus) est la suivante :

UpArrow ou **DnArrow** – monter ou descendre d'une ligne et faire défiler les extrêmes.

CrtlUpArrow ou **CrtlDnArrow** – se déplacer vers le haut ou le bas de la page.

PgUp ou **PgDn** – monter ou descendre d'une page.

CrtlPgUp ou **CrtlPgDn** – monter ou descendre de 1/64ème de la plage de défilement (défilement assez rapide).

ShftCrtlPgUp ou **ShftCrtlPgDn** – monter ou descendre de 1/16ème de la plage de défilement (défilement très rapide).

CrtlHome ou **CrtlEnd** – aller au début ou à la fin de la plage de la barre de défilement.

3.2.6.6 Rafraîchissement des inspecteurs

Le contenu des inspecteurs est actualisé immédiatement au terme de chaque pas. Si vous souhaitez actualiser les inspecteurs à un autre moment, utilisez l'élément de menu "inspect, refresh all panes/inspectors" ("inspecter, actualiser tous les volets/inspecteurs"). Vous pouvez également utiliser cet élément de menu pour vous assurer que les zones de mémoire observées par les inspecteurs de données sont à jour. Certains d'entre eux peuvent changer leur adresse.

3.2.6.7 Répétition d'un inspecteur dans une nouvelle session

Vous pouvez conserver vos inspecteurs dans une nouvelle session de débogage en les laissant tout simplement ouverts lorsque vous utilisez l'élément de menu "file, restart debuggee". Si vous démarrez ensuite une nouvelle session avec le même débogueur, le titre de l'inspecteur apparaîtra dans le menu. Voir la section sessions. GoBug se souvient également (dans une liste déroulante) des 10 dernières adresses dans les inspecteurs de données et de code si vous les avez spécifiquement saisies.

3.2.6.8 Vidage des inspecteurs

Assurez-vous que l'inspecteur et les données que vous souhaitez sauvegarder sont visibles et accédez à l'élément de menu "file, dump" ("Fichier, vidage"). Cliquez sur l'inspecteur approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier destine à accueillir le vidage (GoBug en proposera un). Choisissez également un chemin d'accès pour que ce même fichier (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer ; utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains. Si vous supprimez un inspecteur de données ou de code, vous pouvez modifier le point de départ et la taille du vidage. Même si l'inspecteur a été maximisé, le vidage sera limité à la taille de départ de l'inspecteur. Les inspecteurs de symboles sont toujours entièrement vidés. Le fichier créé sur ce vidage est du texte brut avec retours chariot et sauts de ligne mais sans tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur défini sur une police de largeur fixe (par opposition à une police proportionnelle).

3.2.6.9 Impression des inspecteurs

Assurez-vous que l'inspecteur et les données que vous souhaitez imprimer sont visibles et accédez à l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur l'inspecteur approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Si vous imprimez le contenu d'un inspecteur de données ou de code, vous pouvez modifier le point de départ et la taille de l'impression. Les inspecteurs de symboles sont toujours imprimés en entier (le nombre de pages apparaît dans la boîte de dialogue d'impression). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.2.7 Le volet de traçage de la pile (stacktrace pane)

3.2.7.1 Le volet de traçage de la pile

Pour afficher le volet de traçage de pile (stacktrace), cliquez sur "view, stacktrace".

Ce volet se rafraîchit automatiquement après chaque pas (action F5), et si vous cliquez sur "inspect, refresh all panes and inspectors" ("inspecter, actualiser tous les volets et inspecteurs").

Le volet de traçage de pile sera grisé si la pile visualisée appartient à un thread qui n'est pas en pas-à-pas (c'est-à-dire un thread qui a une action F7, F8 ou F9). Dans ce cas, le traçage de pile correspondra au dernier état connu de la pile.

3.2.7.2 Ce que montre le traçage de pile

Le traçage de pile affiche des informations sur les "adresses de retour" présentes sur la pile à l'ESP courant ou à des valeurs plus élevées d'ESP (c'est-à-dire plus profondément dans la pile). Il affiche donc les fonctions (procédures) qui ont été appelées par l'application pour atteindre le point d'exécution courant.

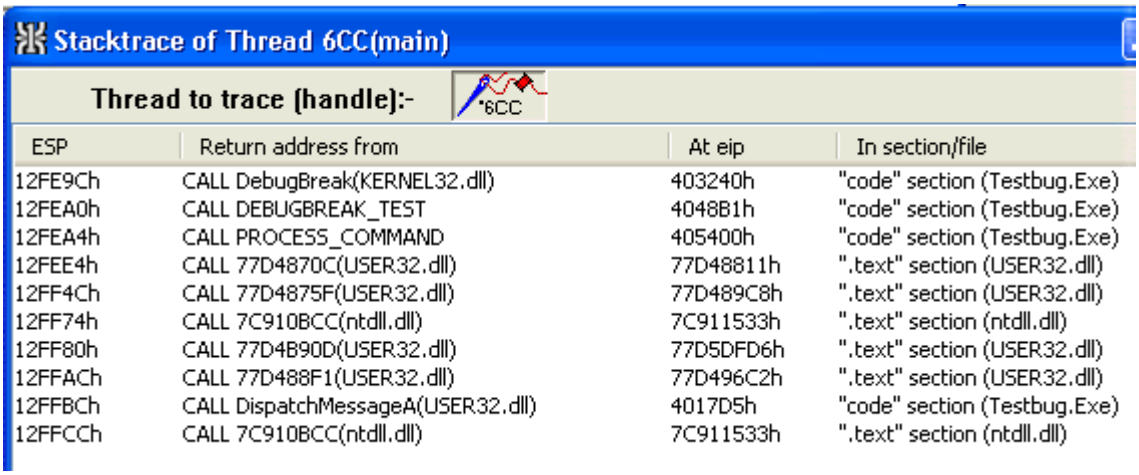
GoBug identifie une adresse comme étant une "adresse de retour" parce qu'immédiatement avant, figurent des opcodes caractérisant un CALL. On le voit bien dans le code qui suit :

```
401020: MOV EAX,EDX
401022: CALL CALCULATE_ASSETS
401027: MOV [ASSETS],EAX          ; met en mémoire le résultat du CALL qui précède
```

Ici GoBug sait que 401027h est une adresse de retour car, 5 octets avant cet emplacement, une instruction CALL a été identifiée.

Le traçage de pile ignore tout ce qui se trouve sur la pile sur les valeurs plus basses de ESP (donc plus hautes sur la pile). En effet, les informations qui s'y rapportent ne concernent que l'exécution passée et ne sont pas pertinentes pour le traçage de pile courant.

Voici le traçage de pile typique d'un thread en pas-à-pas dans la boucle de débogage :



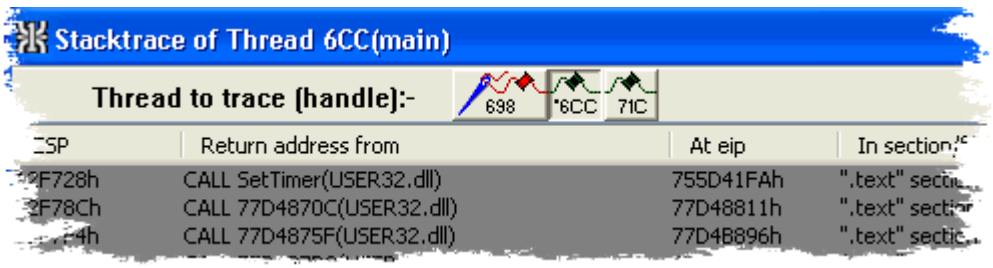
ESP	Return address from	At eip	In section/file
12FE9Ch	CALL DebugBreak(KERNEL32.dll)	403240h	"code" section (Testbug.Exe)
12FEA0h	CALL DEBUGBREAK_TEST	4048B1h	"code" section (Testbug.Exe)
12FEA4h	CALL PROCESS_COMMAND	405400h	"code" section (Testbug.Exe)
12FEE4h	CALL 77D4870C(USER32.dll)	77D48811h	".text" section (USER32.dll)
12FF4Ch	CALL 77D4875F(USER32.dll)	77D489C8h	".text" section (USER32.dll)
12FF74h	CALL 7C910BCC(ntdll.dll)	7C911533h	".text" section (ntdll.dll)
12FF80h	CALL 77D4B90D(USER32.dll)	77D5DFD6h	".text" section (USER32.dll)
12FFACh	CALL 77D488F1(USER32.dll)	77D496C2h	".text" section (USER32.dll)
12FFBCh	CALL DispatchMessageA(USER32.dll)	4017D5h	"code" section (Testbug.Exe)
12FFCCh	CALL 7C910BCC(ntdll.dll)	7C911533h	".text" section (ntdll.dll)

Les colonnes affichent les informations suivantes :

- **Première colonne ("ESP")** – affiche la position sur la pile qui contient "l'adresse de retour" considérée.
- **Deuxième colonne ("Return address from")** – contient un désassemblage de l'instruction CALL dans le code précédant immédiatement l'adresse de retour considérée. Dans le cas d'un CALL à un point dans le debuggee ou l'une de ses DLL, si des symboles sont chargés, le symbole de code peut être affiché. Si l'adresse linéaire du CALL n'est pas reconnue comme correspondant à un symbole particulier, cette adresse est affichée avec le module qui contient cette adresse entre parenthèses. Dans le cas d'un appel à une API Windows, ces symboles seront reconnus car ils sont trouvés en tant qu'exports dans la DLL Windows.
- **Troisième colonne ("at EIP")** – affiche l'adresse de l'instruction CALL qui est désassemblée dans la deuxième colonne.
- **Quatrième colonne ("in section/file")** –montre la section et le module exécutable contenant l'adresse de l'instruction CALL qui est désassemblée dans la deuxième colonne.

3.2.7.3 Le traçage de pile dans les applications multi-thread

Chaque thread possède son propre espace de pile. Dans les applications multithread vous pouvez donc aller d'un thread à l'autre et afficher le traçage de pile en manoeuvrant les boutons de thread correspondants.



Dans le traçage ci-dessus, il y a trois threads : l'un (698h) est en mode pas-à-pas (figuré par la bobine de couleur rouge) dans la boucle de débogage (symbolisée par l'aiguille) alors que les autres fonctionnent normalement (leur bobine est de couleur noire). Le traçage de pile affiché est celui correspondant au handle du thread 6CCh

qui est le thread principal (indiqué par l'astérisque). Si vous visualisez le traçage de pile d'un thread en cours d'exécution, vous obtenez la dernière trace de pile connue de Windows. Étant donné que la pile d'un thread en cours change tout le temps, il s'agira très probablement de la pile lors d'un appel récent à une API Windows.

3.2.7.4 Utilisation du traçage de pile en débogage

La sortie de stacktrace est très intéressante lorsque vous évoluez en pas-à-pas dans votre code, mais son but principal est de vous aider à identifier la zone de code incriminée si votre programme provoque une exception.

Parfois, la cause d'une exception est facile à trouver une fois que vous avez identifié la valeur de EIP (pointeur d'instruction) de l'endroit où l'exception se produit. Mais souvent, l'exception se produit dans une procédure qui est appelée à partir de nombreuses parties différentes de votre programme. Sans la pile, il est très difficile de déterminer quelle partie de votre programme a appelé la procédure. Le traçage de pile le confirmera. La boîte de dialogue d'exception (qui apparaît lorsqu'une exception se produit) contient un bouton permettant d'afficher le traçage de pile.

3.2.7.5 Vidage du traçage de pile

Assurez-vous que la trace de pile est visible et accédez à l'élément de menu "file, dump" ("fichier, vidage"). Cliquez sur stacktrace dans l'élément de menu. Sinon, faites un clic droit sur la pile et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom). Choisissez-lui également un chemin (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains dans vos symboles de code). Le fichier créé sur cette image est du texte brut avec des retours chariot et des sauts de ligne mais ne comporte pas de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur défini sur une police de largeur fixe (par opposition à une police proportionnelle).

3.2.7.6 Impression du traçage de pile

Assurez-vous que la trace de pile est visible et accédez à l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur stacktrace dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.2.8 Focus, navigation et sélection

Cette page décrit la manière selon laquelle GoBug organise le focus et l'activation dans ses fenêtres et volets, et la façon dont vous pouvez utiliser le clavier à la place de la souris pour exécuter les fonctions de GoBug. *Navigation* désigne l'action qui correspond à un changement de focus. *Sélection* signifie le choix d'un élément de menu ou le changement d'état d'un contrôle.

3.2.8.1 Focus sur GoBug et activation

Vous pouvez voir quelle fenêtre de GoBug a le focus du clavier à un moment donné depuis la barre de titre de la manière habituelle. Cependant, puisque vous pouvez avoir plusieurs fenêtres de GoBug ouvertes à un même moment, il y a des choses spéciales que vous pouvez faire. Vous pouvez déplacer le focus d'unee fenêtre de GoBug à l'autre en utilisant CrtlTab (avance) ou ShftCrtlTab (recul). Cela vous permettra de vous déplacer à l'aide du clavier de la fenêtre principale de GoBug aux volets de registres FPU/MMX, inspecteurs et boîtes de dialogue no-modales.

AltF6 bascule le focus entre les deux dernières fenêtres ayant le focus.

De plus, si vous appuyez sur l'une des touches de raccourci, vous pouvez rapidement restaurer le focus dans la fenêtre principale de GoBug et effectuer l'action représentée par la touche de raccourci en même temps, même si une fenêtre secondaire a le focus à ce moment. Vous pouvez également utiliser F10 depuis une fenêtre secondaire pour restaurer rapidement le focus dans la fenêtre principale.

3.2.8.2 Interface clavier système dans les fenêtres de GoBug

F1 – aide

Appui et relâchement de **Alt** (ou **F10**) – activation du menu s'il y en a un (F10 à partir d'une fenêtre secondaire donne le focus à la fenêtre principale de GoBug)

Alt+barre d'espace – affiche le menu du système pour la fenêtre (pour minimiser, déplacer, dimensionner, fermer, restaurer)

AltF4 – Ferme la fenêtre ou GoBug si le focus est sur la fenêtre principale

AltF6 – bascule entre les deux dernières fenêtres actives

Alt+Tab – change pour une application ou une fenêtre active différentes

Alt+Esc – permet de basculer entre applications

3.2.8.3 Navigation autour des volets dans la fenêtre principale de GoBug

Le volet de la fenêtre principale ayant le focus clavier est identifié comme tel par le rectangle de focus dans le volet (un rectangle avec des lignes pointillées). Dans le volet de code et le volet de registres, le fait de déplacer le curseur de la souris entraînera l'apparition d'une surbrillance gris clair à la place du rectangle de focus. Quand un volet a le focus, il peut subir un défilement en utilisant les touches suivantes :

Flèches Up et **Dn** – déplace le rectangle de focus d'une ligne et défile d'une ligne aux extrêmes.

Flèches CrtlUp et **CrtlDn** – aller à l'extrémité de la page.

CrtlHome et **CrtlEnd** – aller en haut ou en bas de la plage de défilement.

PgUp et **PgDn** – fait défiler une page.

CrtlPgUp et **CrtlPgDn** – défilement par 1/64ème de la plage de défilement.

ShftCrtlPgUp et **ShftCrtlPgDn** – défilement par 1/16 ^{ème} de la plage de défilement.

Dans le panneau de code, les flèches Droite et Gauche peuvent étendre le volet si le matériau n'est pas visible.

Dans le volet de registres, les touches fléchées peuvent être utilisées pour naviguer dans les registres et les flags. Appuyez sur Entrée pour modifier un registre ou un flag.

Dans le volet de journal, les touches fléchées afficheront et supprimeront les sous-volets de journal. Voir la section Utilisation du clavier dans le journal

3.2.8.4 Navigation et sélection dans le menu

Utilisation des touches de flèche

Pour ouvrir un menu dans une fenêtre qui a le focus, appuyez et relâchez la touche Alt pour activer la barre de menu puis accédez au menu correct en utilisant les touches de flèche gauche et droite, puis ouvrez-la en utilisant la touche de flèche vers le bas. Naviguez autour du menu en utilisant les touches fléchées vers le haut et la gauche. Utilisez Entrée pour sélectionner un élément de menu et Echap pour ignorer un menu. Vous pouvez également utiliser la touche F10 au lieu de la touche Alt pour activer la barre de menu.

Utilisation de la combinaison Alt+alphanumérique

Appuyez sur la touche Alt puis sur la touche alphanumérique correspondant au caractère souligné pour le menu souhaité. Avant de relâcher la touche Alt, naviguez dans le menu et sélectionnez l'élément de menu de la même manière.

Utilisation des raccourcis de menu

C'est une manière encore plus rapide d'effectuer votre action sans la souris. Utilisez l'un des raccourcis suivants :

F1 – Aide

F5 – Simple/pas (trace dans)

CrtlF5 – Animation en un seul pas

F6 – Simple/pas (sauter par-dessus)

F7 – Exécuter / trap / hook / journal complet

F8 – Exécuter / hook / journal partiel

F9 – Exécuter en arrière–plan

CrtlA – Arriver à .. n'importe quel message

CrtlC – Arriver à .. adresse de code

CrtlM – Arriver à .. message

CrtlN – Arriver à .. nouveau sujet

CrtlP – Arriver à .. procédure

CrtlR – Arriver à .. RET suivant

CrtlT – Contrôle du thread (feux de circulation)

3.2.8.5 Navigation et sélection dans les dialogues

Les règles de Windows habituelles en la matière s'appliquent comme suit :

- Utilisez Alt-F4 pour supprimer la boîte de dialogue et Entrée pour "pousser" le bouton par défaut s'il y en a un (bouton fortement souligné).

- Utilisez TAB et shift-TAB pour déplacer le focus d'un contrôle à l'autre. Un rectangle en pointillé autour du contrôle ou de son texte indiquera quel contrôle a le focus.
- Utilisez les touches de flèches pour déplacer le focus dans le groupe de contrôles.
- Pour les boutons, appuyez sur Entrée ou sur la barre d'espace pour "appuyer" sur le bouton qui a le focus.
- Pour les boutons de commande, tels que le bouton "extract" dans la boîte de dialogue d'inspection des ressources, appuyez sur Entrée ou sur la barre d'espace pour afficher le menu contextuel. Naviguez dans le menu en utilisant les touches fléchées, utilisez Entrée pour sélectionner ou Echap pour annuler.
- Pour les boutons-radio (radiobuttons), utilisez les touches fléchées du groupe de boutons radio pour sélectionner un bouton différent.
- Pour les cases à cocher (checkboxes), utilisez la barre d'espace pour en basculer l'état lorsqu'elles ont le focus.
- Pour les zones de liste déroulante (combo boxes), lorsqu'elles ont le focus, appuyez sur la flèche vers le bas pour ouvrir la boîte, déplacer le contenu de la boîte en utilisant les touches fléchées, puis appuyez sur Entrée lorsque l'élément correct a été trouvé. Appuyez sur Echap pour fermer la boîte sans autre action.
- Pour les zones de liste (listboxes), utilisez les touches fléchées pour vous déplacer vers le haut et vers le bas dans la liste et appuyez sur Entrée lorsque l'élément correct a été trouvé. Vous pouvez également utiliser les touches Home, PgUp, PgDn et Fin pour vous déplacer dans la liste.
- Pour les listes (listviews), parcourez la liste en utilisant les flèches haut et bas ; développer et contracter la liste en utilisant les touches droite et gauche.
- Pour les contrôles up/down (spin boxes), utilisez la flèche vers le haut ou vers le bas lorsque le contrôle a le focus.

3.2.8.6 Navigation et sélection dans les feuilles de propriétés

Les règles Windows habituelles s'appliquent comme suit :

- Pour naviguer entre les feuilles de propriétés, utilisez CrtlEsc ou ShftCrtlEsc.
- Alternativement, lorsqu'un onglet de feuille de propriétés bénéficie du focus, utilisez les touches fléchées gauche et droite.
- Utilisez les touches Tabulation ou ShftTab pour naviguer dans les contrôles d'une feuille de propriétés et dans les touches "OK" et "Annuler" et la tabulation.
- Naviguez et sélectionnez les contrôles de la manière habituelle comme pour les boîtes de dialogue. Voir la section navigation et la sélection dans les dialogues.

3.2.8.7 Navigation au clavier et défilement dans différentes fenêtres

Utilisation du clavier dans le journal

Au lieu d'utiliser la souris pour effectuer des actions dans le journal, vous pouvez utiliser les touches ou combinaisons de touches suivantes :

TAB – si le volet de journal fait partie intégrante de la fenêtre principale, déplacez le focus sur le volet de journal principal

CrtlTAB – si le volet de journal est séparé, déplacez le focus sur le volet de journal à partir d'autres volets et fenêtres

TAB – déplace le focus entre volets de journal ouverts (vers l'avant)

ShftTAB – déplace le focus entre les volets de journal ouverts (vers l'arrière)

UpArrow et **DnArrow** – déplace le rectangle de focus vers le haut et vers le bas sur les éléments du volet de journal qui a actuellement le focus

CrtlUpArrow et **CrtlDnArrow** – déplace le rectangle de focus sur la première ou la dernière ligne

RtArrow – ouvre le volet de journal si le bouton "plus" est rouge, ou affiche les valeurs du registre si le bouton "R" est rouge

LeftArrow – ferme le volet de journal en cours de visualisation ou ferme le volet de valeur du registre s'il est affiché

PgUp et **PgDn** – fait défiler vers le haut et vers le bas le volet de journal s'il y a une barre de défilement

CrtlPgUp et **CrtlPgDn** – fait défiler rapidement le volet de journal s'il y a une barre de défilement

Défilement des inspecteurs (scrolling)

À l'aide de la souris, vous pouvez provoquer un défilement en cliquant sur les flèches de la barre de défilement ou sur la barre de défilement ou en faisant glisser la boîte de défilement. L'interface du clavier (lorsque l'inspecteur a le focus) est la suivante:

UpArrow et **DnArrow** – monte ou descend d'une ligne tout en faisant défiler les extrêmes.

CrtlUpArrow et **CrtlDnArrow** – se déplace vers le haut ou le bas de la page.

PgUp et **PgDn** – monte ou descend d'une page.

CrtlPgUp et **CrtlPgDn** – monte ou descend de 1/64ème de la plage de défilement (défilement assez rapide).

ShftCrtlPgUp et **ShftCrtlPgDn** – monte ou descend de 1/16ème de la plage de défilement (défilement très rapide).

CrtlHome et **CrtlEnd** – accède au début ou à la fin de la plage de la barre de défilement.

Défilement de la boîte de dialogue d'informations

À l'aide de la souris, vous pouvez provoquer un défilement en cliquant sur les flèches de la barre de défilement. L'interface du clavier (lorsque la boîte de dialogue d'information a le focus) a les fonctionnalités suivantes :

UpArrow et **DnArrow** – monte ou descend d'une ligne.

PgUp et **PgDn** – monte ou descend d'une page.

Home et **End** – accède au début ou à la fin de l'information.

Utilisation du clavier dans l'inspecteur de symboles

Au lieu d'utiliser la souris pour effectuer des actions dans l'inspecteur de symboles, vous pouvez parvenir aux mêmes résultats avec les touches ou combinaisons de touches suivantes :

a – ajout d'un élément

d – suppression d'un élément en surbrillance

TAB – met en surbrillance un élément visible

ShftTAB – supprime la surbrillance

Voir aussi défilement des inspecteurs.

Interface clavier dans la boîte de dialogue de raccourci clavier

Tab, Esc, Enter et les touches de flèches sont réservées pour vous permettre de naviguer et de sélectionner dans la boîte de dialogue de la manière habituelle, mais les combinaisons Shift, Crtl et Alt de ces touches peuvent être utilisées comme touches de raccourci.

Utilisation du clavier dans la boîte de dialogue des ressources

L'interface clavier dans la boîte de dialogue des ressources est la même que pour les autres boîtes sauf que vous pouvez afficher une ressource bitmap en sélectionnant la ligne contenant le nom du bitmap en utilisant les flèches haut et bas lorsque le focus est dans le contrôle listview. Puis, utilisez la flèche droite pour faire apparaître l'image bitmap et la flèche gauche pour l'enlever. Utilisez les flèches haut et bas pour consulter les autres bitmaps.

Utilisation de la molette de la souris

GoBug prend en charge le logiciel Intellimouse pour les systèmes Windows 95 et prend également en charge l'utilisation de la molette de souris sur les versions ultérieures de Windows. Si la molette de la souris est configurée pour envoyer un certain nombre de lignes de défilement par mouvement de roue, alors la fenêtre *qui contient le curseur* défilera lors de l'utilisation de la molette de la souris. Si cette dernière est configurée pour un défilement d'une page par mouvement de molette de la souris, Windows envoie une commande de touche PgUp ou PgDn. Pour cette raison, lorsque vous utilisez ce paramètre, c'est la *fenêtre avec le focus* qui défilera à la place de la fenêtre contenant le curseur.

3.3 Utilisation de GoBug

3.3.1 Démarrage, redémarrage, fermetture, changement de debuggee

3.3.1.1 Démarrage d'un debuggee

La classique boîte de dialogue d'ouverture de fichier est la première chose que vous voyez apparaître lorsque vous démarrez GoBug. Elle vous permet de trouver le debuggee que vous voulez démarrer avec le débogueur. Vous ne pouvez démarrer que des fichiers EXE et pas des DLL car le code de ces dernières ne pouvant être appelé que par un processus déjà en cours d'exécution. Si vous avez utilisé GoBug avant les dix derniers programmes que vous avez commencé à utiliser, GoBug apparaîtra dans la liste déroulante.

3.3.1.2 Démarrage d'un debuggee par la ligne de commande

Vous pouvez éviter le passage par cette boîte de dialogue d'ouverture de fichier ouvert en choisissant de spécifier le débuggee directement dans la ligne de commande de GoBug. Cela vous permet de commencer le déboguage plus rapidement. Evidemment, rien ne vous empêche, dès lors, de créer un fichier batch contenant la ligne de commande GoBug et l'exécuter à partir d'une fenêtre MS-DOS.

Utilisez la syntaxe suivante dans la ligne de commande :

```
GoBug path+filename [parameters]
```

Dans cette configuration, vous fournissez le chemin d'accès et le nom de fichier du debuggee. Si vous ne précisez pas une extension, l'extension "exe" est présumée. Fournir des paramètres est facultatif mais s'ils sont néanmoins fournis, ils sont supposés être des paramètres du debuggee, de sorte que vous puissiez voir la réponse du débogueur à certaines options de la ligne de commande.

3.3.1.3 Aller à l'adresse de départ

Quand un programme est lancé par GoBug, il est autorisé à s'exécuter à l'adresse de départ. C'est le point d'entrée du code que vous avez spécifié à l'éditeur de liens au moment de la compilation (notez que certaines combinaisons d'assembleur et d'éditeur de liens sont effectuées automatiquement). Lors du chargement du debuggee, le système effectue certains traitements et charge les DLL requises. Cependant, le debuggee en tant que tel n'exécute aucun code. Pour cette raison, GoBug ignore ce processus. Il serait possible de s'arrêter plus tôt, de sorte que le chargement des DLL puisse être surveillé. Faites-moi savoir sur JG@JGnet.co.uk si cela pourrait s'avérer utile.

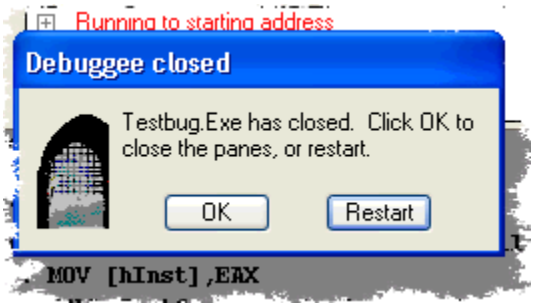
3.3.1.4 Redémarrage du debuggee

Choisir l'option de menu "file, restart debuggee" ("fichier, redémarrer le débogueur") fermera le debuggee qui est actuellement ouvert puis GoBug le rechargera et retournera au point d'entrée et, enfin, s'arrêtera. Ceci est utile si vous voulez revenir au point où une erreur s'est produite.

3.3.1.5 Fermeture du debuggee

Le fait de choisir l'élément de menu "file, close debuggee" ("fichier, fermer le debuggee") fermera le debuggee qui est actuellement ouvert. Vous pouvez donc utiliser le menu Fichier pour ouvrir un nouveau debuggee.

Si le debuggee est en cours d'exécution, il peut également se fermer normalement. Si cela arrive, GoBug affiche la boîte suivante :



Si vous voulez voir le contenu des volets ou inspecter le debuggee avant de fermer les volets de GoBug, vous pouvez le faire. Lorsque vous avez terminé, cliquez sur OK pour fermer les volets de GoBug, ou choisissez "Redémarrer" pour redémarrer le debuggee.

Si vous fermez GoBug pendant le débogage, le debuggee se fermera d'abord, puis GoBug en fera autant.

3.3.1.6 Comment GoBug ferme le debuggee

Dans la mesure où le debuggee a été démarré par GoBug, il peut également être fermé par GoBug. À moins que le debuggee ne se ferme tout seul ou que vous ne le fermiez vous-même, GoBug doit effectuer cette opération. A cet effet, il essaie plusieurs procédés, de manière séquentielle, et les méthodes essayées sont affichées sur la ligne d'état. Après la plupart des sessions de débogage, GoBug peut fermer le débogueur proprement et sans problème. Cependant, il arrive que ces efforts échouent. Par exemple, il se peut que le debuggee ait provoqué une exception. Dans ce cas, GoBug essaiera si nécessaire de sauter par-dessus l'exception pour qu'elle ne se reproduise pas pendant le processus de fermeture. Cependant, cela pourrait causer d'autres problèmes pour le debuggee. Ou il se peut même que le debuggee soit pris dans une boucle continue affectant le processus de messagerie. Si le debuggee a plus d'un thread, il est important que tous les threads soient fermés. GoBug essaie les méthodes suivantes pour fermer le debuggee aussi proprement que possible : $$$

1. Envoi de WM_QUIT aux threads du debuggee. Cela fonctionne si le debuggee a établi (ou établit automatiquement) une file d'attente de messages à moins que le processus de messagerie n'ait échoué.
2. Enjoint au debuggee d'appeler *ExitProcess*. Ceci est fait en détournant l'exécution du debuggee vers cette API et est utilisé si le debuggee n'a pas de file d'attente de messages ou si WM_QUIT a échoué.
3. Fermeture du créateur. Ici, le thread GoBug qui a créé le debuggee est fermé, ce qui indique au système de fermer tous les processus enfants.
4. Appel de *TerminateProcess*. Ce n'est pas une méthode privilégiée car le système est incapable d'informer les DLL éventuellement chargées dans l'espace d'adressage du debuggee. De même, si les précédentes tentatives de fermeture ont échoué, il est tout à fait possible qu'il y ait de la corruption. Pour cette raison, GoBug vous avertit d'éventuels problèmes par une boîte de message.

Dans un programme multithread, GoBug peut prendre plus de temps pour fermer le que dans un programme mono-thread, car chaque thread doit être traité.

3.3.1.7 Changement de debuggee

Choisir l'élément de menu "file, change debuggee" ("fichier, changement de debuggee") fermera le debuggee qui est actuellement ouvert et vous permettra d'en choisir un autre (ou le même à nouveau) en utilisant la boîte de dialogue d'ouverture de fichier.

3.3.1.8 Ajout de paramètres de ligne de commande au debuggee

Vous pouvez ajouter ces paramètres dans la boîte de dialogue d'ouverture de fichier. Cela peut être utile si, par exemple, vous voulez tester la réponse du debuggee à certaines options de ligne de commande. Vous pouvez également ajouter des paramètres de débogage si vous spécifiez le debuggee dans la propre ligne de commande de GoBug, en court-circuitant la boîte de dialogue d'ouverture de fichier. Voir le paragraphe Démarrage d'un debuggee à l'aide de la ligne de commande.

3.3.1.9 "Sessions" du debuggee

Gobug conserve un enregistrement (dans le fichier ini de GoBug) des dix dernières "sessions" de débogage, y compris les points de break saisis et les données, codes et symboles inspectés par l'utilisateur pendant la session. Une "session" est spécifique au débogage, donc si vous déboguez à nouveau le même programme, ces enregistrements apparaissent comme des options dans les éléments du menu. Notez que la modification des paramètres de ligne de commande pour un programme particulier entraînera l'enregistrement d'une nouvelle session. Cependant, recompiler ou lier à nouveau le programme ne le fera pas. Cela signifie que vous pouvez essayer de corriger une erreur et ensuite trouver facilement le même point d'arrêt que précédemment. L'enregistrement d'un point d'arrêt dans le fichier ini est (si possible) conservé par nom de symbole plutôt que par adresse, de sorte que le point d'arrêt sera toujours disponible après recompilation tant que le nom du symbole n'est pas modifié. Toutefois, lorsque l'enregistrement est conservé par *adresse*, il peut ne pas être utile si l'adresse a été déplacée dans le processus de recompilation.

3.3.2 Reprise du contrôle sur les threads récalcitrants

3.3.2.1 Le thread rebelle

Lors du débogage, un thread peut devenir rebelle et capricieux de la façon suivante :

- S'il n'est pas encore revenu d'un CALL. Mais, comment le savoir ? Eh bien, le thread ne sera pas dans la boucle de débogage et la flèche EIP (montrant le lieu de l'instruction en cours) pointera sur le CALL ou immédiatement après. La flèche EIP s'animera (ou éventuellement une boîte de message apparaîtra) si vous appuyez sur une touche d'action. Comment cela peut-il arriver? En général, c'est parce que l'API attend une action supplémentaire dans le debuggee. Par exemple, dans une boucle de message, l'API *GetMessage* ne retournera pas tant que l'utilisateur ne fournira pas une entrée au debuggee. Cette entrée attendue pourrait être un simple mouvement de la souris sur la fenêtre du débogué, un clic de souris ou une pression sur une touche. Pour prendre un autre exemple, il se pourrait également que le CALL ait créé une *boîte de dialogue modale*, auquel cas il ne retournera pas tant que la boîte de dialogue n'est pas fermée. Il se peut aussi que l'appel attende qu'un événement se produise dans un autre thread. Enfin, une autre possibilité serait que le code est dans une boucle continue dans le CALL. Reportez-vous, sur ce point, à la section Comment gérer l'échec de retour après un CALL.
- Voir également Reprendre le contrôle d'un thread dans une boucle continue.
- Le flag de trap a été libéré. Ce serait le cas si vous aviez appuyé sur F8 ou F9. Les volets seraient gris foncé. Une fois que flag de trap a été libéré, cela signifie que les instructions du thread ne viennent plus dans la boucle de débogage pour l'analyse et le rapport.

- Il y a une boucle continue. Si cela se produit dans le thread principal du debuggee, il sera incapable de peindre l'une de ses fenêtres. Il apparaîtra que le débogueur a "gelé", même quand il est en cours d'exécution.

3.3.2.2 Action si un thread n'est pas retourné à partir d'un CALL

Faites en sorte que le CALL retourne en effectuant l'action attendue. Si vous attendez dans *GetMessage*, déplacez la souris sur une fenêtre du debuggee ou cliquez sur la fenêtre. Si vous attendez dans une boîte de dialogue modale, fermez-la. Si vous attendez que quelque chose se produise dans un thread, faites que ce thread s'exécute. L'utilisation de la touche de raccourci va simuler un message et entraînera un retour de *GetMessage*, tout comme le contrôle du feu de circulation. Si vous pensez que la raison pour laquelle le CALL n'est pas retourné tient au fait qu'il est bloqué dans une boucle continue, utilisez la touche de raccourci ou le contrôle du feu de circulation.

3.3.2.3 Reprise du contrôle d'un thread si le flag de trap a été libéré

Utilisez le raccourci clavier ou le contrôle de feu tricolore. Dans le contrôle de feu tricolore, réduisez l'action pour essayer d'activer le flag de trap (action F5, F6 ou F7) et cliquez sur "apply". Comme GoBug ne sait pas où le thread en cours s'exécute, il inonde le code avec des INT 3, provoquant ainsi une entrée dans la boucle de débogage, et le thread est alors intercepté. Notez cependant que cela ne fonctionne pas toujours !

3.3.2.4 Reprise du contrôle d'un thread dans une boucle continue

Si l'action en cours est F7, alors le flag de trap n'a pas été libéré. Il suffit d'appuyer sur F5 ou F6 pour reprendre le contrôle du thread. Sinon, si le flag de trap a été libéré, utilisez le raccourci clavier ou le contrôle de feu tricolore. Dans le contrôle du feu, réduisez l'action pour essayer d'activer le flag de trap (action F5, F6 ou F7) et cliquez sur "apply". Comme GoBug ne sait pas où le thread en cours s'exécute, il inonde le code avec des INT 3, provoquant ainsi une entrée dans la boucle de débogage, et le thread est alors intercepté. Notez cependant que cela ne fonctionne pas toujours !

3.3.3 Pas-à-pas, traçage, saut et fonctionnement

3.3.3.1 Pas-à-pas (traçage dans) (F5)

Sous réserve que le volet de code soit "actif" (c'est-à-dire non-grisé), et que GoBug ait le focus sur le clavier, vous pouvez faire du pas-à-pas en appuyant sur F5. Le pas-à-pas effectue, à chaque fois et comme son nom l'indique, une seule instruction du processeur. Les résultats apparaissent dans les volets de GoBug. La position courante du registre EIP, c'est-à-dire le lieu de l'instruction courante, est indiqué dans le volet de code (grande flèche). Si F5 est pressé lors d'une instruction "CALL", le traitement continuera à la procédure correspondante. C'est-à-dire que le lieu d'instruction va se déplacer dans le CALL. Si F5 est pressée sur une instruction "JMP", le traitement se poursuivra à la destination du saut. Si F5 est manœuvrée sur une instruction de saut conditionnel (par exemple, JZ, JNZ ou JC), alors le traitement va sauter ou non sur la destination du saut, selon l'état des flags. La flèche du pointeur EIP changera de direction et de couleur pour montrer ce qui va se passer. Lors d'un simple pas, GoBug alimente un journal complet des événements, des messages, des instructions exécutées et des valeurs de registre constatées. Les événements et les messages sont consignés à l'aide d'un hook qui est inséré dans le contexte du debuggee. Les instructions exécutées sont consignées lorsque le debuggee entre dans la boucle de débogage. Vous avez ensuite toute latitude pour consulter le journal et y trouver un historique des actions du debuggee.

3.3.3.2 Pas-à-pas dans le code de Windows

En appuyant sur F5 lorsque l'instruction est à un CALL d'API Windows, des problèmes peuvent survenir lorsque vous entrez dans le code de l'API. Dans Windows 9x et ME, il existe un morceau de code spécial qui rejette tout débogueur essayant de faire ne serait-ce qu'un pas dans le code. GoBug contourne ce bout de code (si F5 est enfoncé) en le remplaçant à l'endroit approprié. Cependant, force est de constater que certaines DLL Windows 9x et ME vérifient en permanence si elles sont en cours de débogage. Si tel est le cas, elles rejettent le débogueur, généralement en appelant *ExitProcess* au nom du debuggee, le terminant ainsi. En plus de cela, certaines DLLs traînent encore du code Win16. Celui-ci sera exécuté, mais n'apparaîtra pas correctement dans le volet de code de GoBug. En raison de ces difficultés dans Windows 9x et ME GoBug affiche une boîte de dialogue d'avertissement avant d'entrer dans une DLL système.

Windows NT/2000/XP n'ont pas les bouts de code spéciaux mentionnés plus haut et généralement il n'y a pas de problème à faire du pas-à-pas à travers le code du système sur ces plates-formes.

Cependant, toutes les DLL de Windows sont écrites dans un langage de haut niveau (et non en assembleur), de sorte qu'il y a beaucoup de «bagages» et de code inutile dans ces DLL, ce qui les rend très difficiles à lire. Donc, même si vous pouvez faire un pas à travers le code de Windows, l'exercice se révèlera finalement inutile. GoBug est conçu pour déboguer votre code pas celui des autres.

3.3.3.3 Pas-à-pas et animation

En appuyant sur Crtl et F5, le processus "s'anime". Cela équivaut à appuyer sur F5 (après un léger retard) chaque fois que le débogueur arrive dans la boucle de débogage. Tous les CALL sont tracés, sauf pour les API. Animate est utile si vous voulez suivre une progression rapide jusqu'au point d'erreur dans votre programme. Le fait de cliquer n'importe où dans la fenêtre principale ou d'appuyer sur n'importe quelle touche lorsque GoBug a le focus arrêtera l'animation.

3.3.3.4 Pas-à-pas (saut par-dessus) (F6)

Sous réserve que le volet de code soit "actif" (c'est-à-dire n'apparaissant pas grisé), et que GoBug ait le focus sur le clavier, vous pouvez passer par-dessus un CALL et d'autres instructions en appuyant sur F6. Lorsque vous sautez par-dessus une instruction CALL, vous ne faites pas un seul pas dans la procédure appelée, mais continuez à en faire un pas de l'autre côté. La procédure appelée sera exécutée dans son intégralité, cependant. Vous serez généralement informé dans le volet de code et dans le journal des erreurs d'API dans la procédure appelée, mais il n'y aura pas de détails précis sur d'éventuelles erreurs. Les exceptions, cependant, arrêteront le debuggee dans le CALL. Lorsque vous sautez par-dessus un CALL, GoBug n'enregistre pas les instructions effectuées à l'intérieur du CALL. Cependant, il enregistre les événements et les messages se produisant à l'intérieur du CALL. Les CALL Windows doivent être sautés en utilisant F6 pour éviter d'entrer dans le code Windows, ce qui peut interrompre le processus de débogage (voir ci-dessus). Les autres instructions qui sont sautées en utilisant F6 sont les instructions de "répétition" qui utilisent REP, REPZ ou REPNZ. Encore une fois, elles sont exécutées en totalité mais ne sont pas enregistrés. Notez que vous ne pouvez pas utiliser F6 pour sauter par-dessus une instruction LOOP. C'est parce que, selon le code, il pourrait y avoir une autre sortie de la boucle. Au lieu de cela, vous pouvez définir un point d'arrêt de code au point de sortie de boucle attendu et appuyer sur F9, ce qui aura le même effet.

3.3.3.5 Run/trap/hook/full log (F7)

Ce mode permet au débogueur d'exécuter des instructions en continu, et il n'est pas nécessaire d'appuyer sur une touche à chaque fois. Le debuggee semble fonctionner normalement, bien que plus lentement que d'habitude. Ce qui se passe ici, c'est que chaque instruction arrive dans la boucle de débogage de GoBug et lorsque l'instruction quitte la boucle, GoBug place le flag "trap" dans le processeur pour que l'instruction suivante revienne à la boucle. Cela permet à chaque instruction d'être entièrement enregistrée dans ce mode. Le hook GoBug est également défini dans le contexte du debuggee, de sorte que tous les événements et messages sont également consignés. Le mode F7 vous permet donc d'exécuter le programme jusqu'à ce qu'une condition d'erreur se produise, puis vous pouvez afficher l'historique du débogueur dans le journal. En mode F7, le code n'est pas suivi dans les DLL système en mémoire partagée. Cela garantit qu'il n'y a aucun problème découlant de l'opposition de Windows à cela. Cependant, parfois, ces DLL ne sont pas chargées dans la mémoire partagée mais dans le propre contexte du debuggee. Dans ces cas, l'action F7 ira dans les DLL, et le debuggee semblera fonctionner beaucoup plus lentement que d'habitude.

3.3.3.6 Run/hook/part log (F8)

Ce mode permet au débogueur d'exécuter des instructions en continu, mais il s'exécutera beaucoup plus rapidement que le mode F7 car le flag de trap est libéré. Les instructions ne sont pas consignées dans le journal, mais les messages et les événements le sont car le hook est défini dans le contexte du débogueur. En mode F8, le débogueur fonctionne plus rapidement qu'en mode F7, mais plus lentement qu'en mode F9. Le mode F8 est utile si vous n'êtes pas intéressé par le journal des instructions, mais que vous avez besoin du journal des messages. Notez que puisque le flag de trap est libéré dans ce mode, il peut être difficile de remettre l'exécution dans la boucle de débogage. Pour ce faire, vous devrez peut-être définir un point d'arrêt, essayer de reprendre le contrôle du thread rebelle ou redémarrer le debuggee.

3.3.3.7 Fonctionnement en arrière-plan (F9)

En mode F9, le débogueur fonctionne si vite que vous ne pouvez pas percevoir qu'il est en cours de débogage. Les instructions ne vont pas dans la boucle de débogage parce que le flag de trap est libéré. Le hook est également libéré donc il n'y a pas de journal des événements et des messages non plus. Laissé dans ce mode, le debuggee fonctionnera normalement jusqu'à ce qu'une exception se produise (quand GoBug prend ensuite le relais) ou jusqu'à la fermeture. Vous utiliseriez le mode F9 si vous vouliez exécuter le débogueur normalement jusqu'à ce qu'une exception se produise. Notez que puisque le flag de trap est libéré dans ce mode, il peut être difficile de remettre l'exécution dans la boucle de débogage. Pour ce faire, vous devrez peut-être définir un point d'arrêt, essayer de reprendre le contrôle du thread rebelle ou redémarrer le debuggee.

3.3.3.8 Utiliser la souris au lieu des touches pour changer l'action

Vous pouvez utiliser le menu "action" pour ce faire.

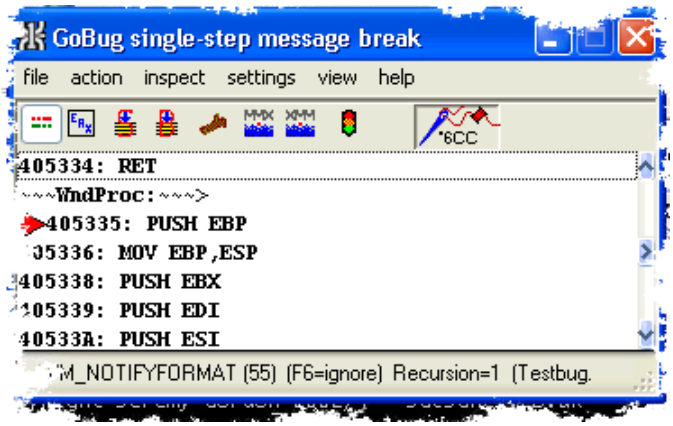
3.3.3.9 Utilisation du contrôle par feux tricolores pour changer l'action

Cliquez sur le bouton de feux tricolores dans la barre d'outils (également accessible à partir du menu "action" ou en appuyant sur Crtl+T), puis cliquez sur les grands boutons radio appropriés. Appuyez sur "Appliquer" pour effectuer le changement ou "Annuler" pour ne rien faire.

3.3.4 Break de message en pas-à-pas

3.3.4.1 Le break de message en pas-à-pas

S'il est activé (à l'état « on ») et que vous faites un pas au travers de votre code, tout message arrivant à une procédure de fenêtre dans le debuggee servira de point d'arrêt. Il provoquera l'entrée du debuggee dans la boucle de débogage au moment où la procédure de fenêtre est appelée avec le message. La poursuite du traitement sera interrompue jusqu'à ce que vous décidiez quoi faire. Vous pouvez voir le message concerné dans la barre d'état.



Ici vous pouvez voir à partir de la barre d'état que le message WM_NOTIFYFORMAT a été envoyé au debuggee, et vous pouvez voir à partir du volet de code qu'il a été envoyé à la procédure de fenêtre "WndProc:", et vous pouvez voir les premières lignes du code la procédure de la fenêtre. Sur la ligne d'état, vous pouvez voir "Recursion = 1". Cela signifie que le code était déjà en cours d'exécution dans la procédure de fenêtre lorsque ce message a été envoyé. La première fois qu'un message est envoyé La récursivité sera 0. Si, avant que le code ne revienne de la procédure de fenêtre, un autre message est envoyé (sans doute causé par quelque chose qui se passe dans la procédure de fenêtre), il sera à 1, etc.

3.3.4.2 Options disponibles lorsque le break se produit

Lorsqu'un break de message en pas-à-pas se produit, vos options sont les suivantes :

- **F5** – ceci vous fait accéder en un pas au code de la procédure de fenêtre et alors, vous pouvez avancer normalement dans ce code.
- **F6** – ceci vous fait sauter entièrement par-dessus le message. Si un autre message suit, il apparaîtra à la place. Cependant, vous pouvez trouver, à la fin d'une série de messages, que vous revenez à la séquence de code principale (qui mène à la boucle de message ou qui s'y trouve).
- **F7**, **F8** ou **F9** – ceci vous fait sauter entièrement par-dessus le message et, puisque vous n'êtes plus en pas-à-pas, aucun autre message ne provoquera un break dans la procédure de fenêtre à moins que vous ne définissiez un point d'arrêt spécifique à cet effet.
- Vous pouvez poursuivre le traitement en définissant un autre point d'arrêt. Comme cela provoquera un "run", aucun break de message en pas-à-pas ne se produira.

3.3.4.3 Effet d'ignorer le message en utilisant F6

Lorsque vous appuyez sur F6 après un break de message en pas-à-pas, le volet de code change pour vous montrer ce qui s'est passé et pendant très peu de temps le message reste dans la ligne d'état. Afin de garder le contrôle, GoBug place le flag de trap sur chaque instruction entrant dans la boucle de débogage, mais n'enregistre aucune instruction, bien qu'il enregistrera les messages qui apparaissent pendant le traitement du message ignoré, voir, à ce sujet récursion du message. Garder le trap ne ralentit généralement pas le debuggee, mais s'il a beaucoup de travail à faire pendant le traitement du message, vous pouvez voir les lignes rouges verticales animées ignorées dans le volet de code. Cela indique que le débogueur est toujours en train de traiter le message dans la procédure de fenêtre et n'a pas encore retourné.

3.3.4.4 Opération par thread du break de message en pas-à-pas

En pratique, généralement, le break de message en pas-à-pas ne fonctionne que sur un thread de votre programme. Cela est dû au fait que les messages Windows sont normalement impliqués dans l'interface utilisateur et qu'il est de mauvaise pratique de programmation que d'avoir plusieurs threads impliqués. Cependant, il est possible qu'un thread secondaire ait sa propre boucle de message ou reçoive des messages. En fait, c'est un bon moyen pour les threads de communiquer les uns avec les autres. Ainsi, GoBug permet-il l'utilisation du break de message en pas-à-pas avec plus d'un thread. N'oubliez pas que chaque fois que le debuggee est dans la boucle de

débogage, *tous* ses threads sont suspendus. Donc, jusqu'à ce que vous autorisiez le processeur à continuer, aucun message ne sera reçu par un autre thread.

3.3.4.5 Activation et désactivation du break de message en pas-à-pas

Pour le thread principal et pour les nouveaux threads, cela peut être fait dans le menu "action" en activant ou désactivant la coche correspondant à l'élément de menu approprié. Pour tous les threads, vous pouvez contrôler le saut de message en une seule étape en utilisant le contrôle de feux tricolores.

3.3.4.6 Messages ignorés par le break de message en pas-à-pas

Les messages destinés aux procédures de fenêtre dans les DLLs système sont ignorés par le break de message en pas-à-pas. Il y en a souvent un grand nombre, en particulier lorsqu'il s'agit de dialogues et de contrôles personnalisés. En ce qui concerne GoBug, les "DLLs système" sont celles qui se trouvent dans le dossier Windows\system ou dans le dossier Windows\WinSxS. Si vos DLLs système sont dans un autre répertoire, merci de me le faire savoir sur jg@jgnet.co.uk.

Le break de message en pas-à-pas ignorera également les messages ignorés par le journal. Pour les modifie cette situation, utilisez l'option de menu "settings, log settings" ("paramètres, paramètres du journal") et l'onglet "ignore".

3.3.5 Auto-activation en fonctionnement

Cela amène la fenêtre principale de debuggee au premier-plan quand elle «s'exécute» pour la première fois soit en général (F7, F8 ou F9 enfoncé), soit lorsqu'elle s'achemine vers un point d'arrêt. Choisissez Auto-activate lors de l'exécution ou hors-exécution en utilisant le menu "action". Si cette option est *désactivée*, alors, quand vous appuyez sur F7, F8 ou F9 ou allez vers un point d'arrêt, la fenêtre de GoBug restera au-dessus de la fenêtre principale du debuggee. Évidemment, cela n'affecte pas les programmes de console puisqu'ils n'ont pas de fenêtre. L'activation ou la désactivation de l'Auto-activate est une question de choix personnel : elle n'affecte pas les procédures de débogage pratiques.

3.3.6 Exceptions provoquées par le debuggee

Le logiciel Testbug.exe propose une série de tests illustrant ce qui se passe lorsque le debuggee provoque une exception (intentionnellement ou non). Le paragraphe « Exceptions » de la section « Démonstrations d'"Utilisation de GoBug" » en détaille le contenu dans le volume 3 de cette série de documents.

3.3.7 Erreurs d'API

3.3.7.1 Qu'est-ce qu'une erreur d'API ?

Une erreur d'API se produit lorsque l'API n'a pas pu effectuer l'action demandée pour une raison ou une autre. Evidemment, l'API agit, par construction, de manière contrôlée.

3.3.7.2 Comment le système notifie une erreur d'API

Le système indique une erreur d'API au moyen du dispositif suivant :

- Implémentation d'une valeur de retour (à partir de l'API) dans le registre EAX. Par convention, si cette valeur est "FALSE" (valeur nulle) cela atteste qu'il y a eu erreur. Parfois, la valeur –1 indique une erreur.
- Un code d'erreur au format dword mémorisé dans la base de données de threads pour le thread dans lequel l'erreur s'est produite. Vous pouvez récupérer ce code en appelant l'API *GetLastError* (à partir du thread où l'erreur est apparue). Le code d'erreur permet d'identifier la cause de l'erreur. Ceci est utile pour les erreurs "attendues", où votre programme doit découvrir la nature d'une erreur, ou si la cause de l'erreur doit être signalée à l'utilisateur. Le code d'erreur dans la base de données de threads n'est généralement pas écrasé en l'absence de nouvelle erreur API. Les codes d'erreur sont tous documentés dans le fichier WinError.h dans le Kit de développement logiciel (SDK) ou dans un document appelé "Codes d'erreur système" sur MSDN.

3.3.7.3 Certaines API pratiquent une notification d'erreur d'API différente

Certaines API n'utilisent pas EAX pour indiquer qu'il y a eu une erreur. Au lieu de cela, elles peuvent mettre le code d'erreur à zéro pour caractériser un succès ou à une valeur différente de zéro pour indiquer un échec. Certaines API mettent à 1 le code d'erreur pour signaler des événements dans l'API plutôt que d'afficher des erreurs. Un exemple d'une telle configuration est ce qui se passe avec *CreateFile* si (quand elle est appelée) le flag dwCreationDistribution est positionné sur la valeur 2h (créer toujours). Si le fichier existe déjà, *CreateFile* mettra à 1 le code d'erreur pour signaler cette particularité, mais elle retournera néanmoins avec un handle de fichier dans EAX, comme d'habitude. Vérifiez toujours avec le SDK comment une API particulière signale ses erreurs.

3.3.7.4 Bonnes pratiques de programmation

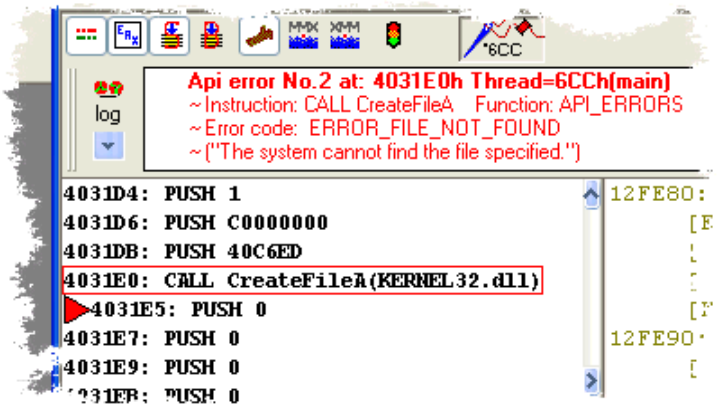
Il est bon et voire souvent essentiel de faire en sorte que votre programme gère les erreurs inattendues qui pourraient survenir lors de l'exécution. De telles erreurs peuvent se produire facilement et pas seulement pendant le développement. Pour prendre un exemple extrême, supposons que vous souhaitiez configurer une zone de mémoire en écriture à l'aide de l'API *VirtualAlloc*. Pour éviter qu'une éventuelle exception ne se produise, vous devez impérativement anticiper un improbable échec d'exécution de cette fonction et tester donc son retour avant toute tentative d'écriture en mémoire. Si *VirtualAlloc* a donné un retour d'erreur, vous devez prévoir une sortie de programme d'allure civilisée et informer l'utilisateur (au moins) que l'action n'a pas pu être effectuée.

Il est également logique, si vous demandez un handle au système, de tester la validité de la valeur de retour dans EAX avant d'utiliser ledit handle. Selon ce principe, vous devriez vérifier EAX si vous avez appelé *CreateFile* pour voir si cette fonction a retourné un bon handle pour le fichier dans lequel vous vous proposez d'écrire (dans le cas présent, EAX = –1 indique un échec).

Et supposons maintenant que vous vouliez vérifier que le handle de fichier est toujours valide après que beaucoup d'autres traitements ont eu lieu avant l'écriture proprement dite dans le fichier. Pour ce faire, il vous est possible d'appeler une API inoffensive telle que *SetFilePointer* avec le flag FILE_CURRENT pour vérifier le handle de fichier. Vous devez ensuite vérifier le retour dans EAX à partir de l'API ou les informations fournies par *GetLastError* pour vous assurer que le handle est toujours valide.

3.3.7.5 Comment GoBug surveille et traite les erreurs d'API

Si vous êtes en pas-à-pas ou que vous effectuez une action F7 (run/trap/full log), puisque le thread qui s'exécute arrive dans la boucle de débogage après chaque instruction, GoBug peut vérifier le dernier code d'erreur du thread pour voir s'il a changé. Si le code d'erreur a changé depuis sa dernière vérification et est supérieur à zéro, et si EAX est égal à zéro ou –1, GoBug prendra conscience qu'une erreur d'API vient de se produire. Dans le volet de code, un rectangle rouge apparaît alors autour de l'appel API incriminé qui avait la dernière erreur pour le thread concerné. Les détails de l'erreur API apparaissent également dans le journal :



Dans cet exemple, le code d'erreur indiqué dans le journal est le nom de la constante provenant du fichier d'en-tête système et la description de l'erreur sur la ligne immédiatement en-dessous est une chaîne correspondant au code d'erreur obtenue à partir de l'API *FormatMessage*.

3.3.7.6 Break sur erreur d'API

Vous pouvez faire en sorte que GoBug s'interrompe sur toute erreur d'API si vous cochez l'entrée correspondante dans le menu "action". Cela ne fonctionne que lorsque vous effectuez une action F7 (run/trap/full log), car le thread qui est en cours d'exécution arrive dans la boucle de débogage après chaque instruction, ce qui permet à GoBug de vérifier le dernier code d'erreur. Si une erreur API se produit alors, GoBug viendra au premier plan et le debuggeee sera suspendu dans la boucle de débogage, prêt à voir ce qui n'a pas fonctionné. C'est un bon moment pour jeter un coup d'œil au traceur de pile pour voir la profondeur d'appel de la partie du code qui indique que l'erreur est survenue. Gardez présent à l'esprit que la partie de la pile contenant les paramètres envoyés à l'API peut avoir été modifiée par l'API elle-même, et peut donc ne pas être fiable. Toutefois, les informations sur les paramètres transmis à l'API seront disponibles à partir du volet de journal d'instructions, à moins que ce ne soit des valeurs conservées en mémoire à ce moment-là. Si l'erreur n'est pas évidente, pour vérifier ces paramètres, redémarrez le débogueur, définissez un point d'arrêt de code avant l'appel de l'API et exécutez la même action dans l'application pour atteindre le même point d'exécution.

3.3.8 Points d'arrêt (Breakpoints)

3.3.8.1 Qu'est-ce qu'un point d'arrêt ?

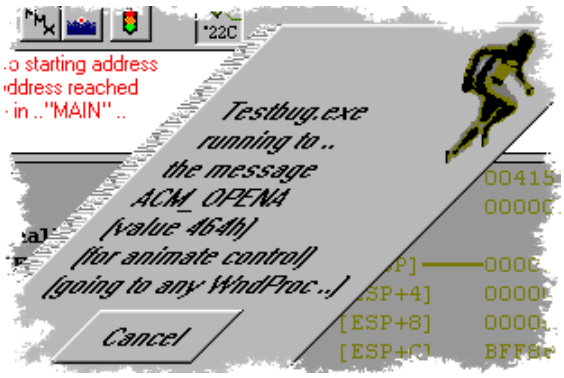
Un point d'arrêt est un endroit du code où l'exécution s'arrêtera et le où le débogueur entrera dans la boucle de débogage. À ce stade, GoBug viendra au premier plan. Si vous voulez examiner l'exécution des instructions de code après ce point, vous pouvez le faire en effectuant tout simplement un seul pas dans le code. La définition et

la mise en place d'un point d'arrêt permettent à votre programme de s'exécuter rapidement jusqu'au point que vous souhaitez examiner.

3.3.8.2 Points d'arrêt d'exécution (run), de code et autres

Un point d'arrêt d'*exécution* provoque l'exécution de tous les threads du debuggee en direction du point d'arrêt. Lorsque vous utilisez un point d'arrêt d'exécution (autre que l'exécution au RET suivant), vous perdez l'opportunité de passer d'un code à l'autre, jusqu'à ce que le point d'arrêt soit atteint. Un point d'arrêt de *code* n'affecte pas, quant à lui, l'action du debuggee puisqu'il est simplement réalisé en insérant une INT 3 statique dans le code. Vous pouvez faire un seul pas jusqu'à ce point ou laisser l'un des threads du debuggee l'atteindre de lui-même. Il existe également d'autres types de points d'arrêt, qui provoquent la rupture automatique du debuggee, comme s'il y avait une exception ou une erreur d'API ou qu'un message se produisait lors d'un simple pas.

3.3.8.3 La fenêtre fantaisie (*Squiffy*)



Lorsque vous définissez un point d'arrêt d'*exécution*, GoBug affiche une fenêtre fantaisie qui donne des détails sur le point d'arrêt. Vous pouvez annuler le point d'arrêt en cliquant sur le bouton "Annuler".

3.3.8.4 Exécution vers le RET suivant

Si vous faites un pas à travers le code et que vous vous retrouvez dans une longue procédure par erreur, cliquez sur l'élément de menu "action, run to .., run to next RET" ("action, exécutez vers .., passez au RET suivant"), ou appuyez sur les touches Crtl+R. Cela provoquera l'exécution à l'instruction RET suivante du thread que vous observez. Bien sûr, cela ne fonctionne que si le debuggee est à la porte d'entrée. Ne l'utilisez pas si vous faites un seul pas depuis l'adresse de départ. Dans ce cas, le code peut ne jamais aller à un RET car il sera probablement en train d'entrer dans la boucle de message ou il peut éventuellement appeler *ExitProcess* sans entrer dans la boucle de message s'il s'agit d'un processus de console. L'exécution de RET ne libère pas le flag de trap. Un journal complet des événements et des messages sur le chemin conduisant au RET est constitué mais il exclut les instructions. De ce fait, il y parvient plus rapidement.

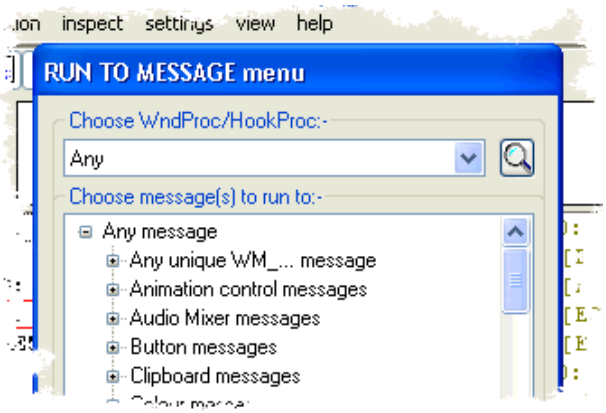
3.3.8.5 Exécution vers un nouveau thread

Ce point d'arrêt est atteint au début du nouveau thread suivant. C'est utile si vous n'êtes pas intéressé par le thread courant mais que vous voulez examiner l'exécution du code par le nouveau thread. Cliquez sur l'élément de menu "action, run to .., run to new thread" ("action, exécutez vers .., exécutez un nouveau thread"), ou appuyez sur les touches Crtl+N. Le flag de trap est libéré et aucun journal d'instructions ou de messages n'est constitué tant que le nouveau thread n'est pas créé.

3.3.8.6 Exécution vers n'importe quel message

C'est un moyen rapide d'observer l'exécution du code dans une procédure de fenêtre. Ce point d'arrêt arrête l'exécution au début de la première procédure de la fenêtre pour recevoir un message (que ce soit directement à partir du système ou via la file d'attente des messages). Cliquez sur l'élément de menu "action, run to .., run to any message" ("action, exécuter à .., exécuter à n'importe quel message"), ou appuyez sur les touches Crtl+A. Le flag de trap est libéré et aucun journal des instructions ou des messages n'est constitué jusqu'à ce que le message se produise. Bien sûr, ce point d'arrêt ne fonctionne que si le debuggee dispose d'une file d'attente de messages et reçoit donc des messages ! Notez que si votre débogueur fonctionne sur une machine alimentée par batterie, il reçoit régulièrement le message WM_POWERBROADCAST, qui sera intercepté par le point d'arrêt. Si vous ne voulez pas déboguer ce message, vous devez être plus précis dans la définition de votre point d'arrêt de message (voir ci-dessous).

3.3.8.7 Exécuter vers message…



Ce point d'arrêt vous permet de spécifier quel message ou quels messages arrêteront l'exécution et, en effet, (si les symboles sont chargés) dans quelle procédure de fenêtre. Cliquez sur l'élément de menu "action, run to .., run to message .. ("action, exécutez pour .., exécutez le message .."), ou appuyez sur les touches Crtl+M pour ce faire.

Les messages sont classés par catégories dans le contrôle Treeview ci-contre. Vous pouvez choisir de pratiquer un break sur des catégories de message ou, en ouvrant la partie pertinente de l'arborescence, vous focaliser sur un seul message. Vous pouvez également entrer le nom d'un message particulier dans le contrôle d'édition de la partie haute de la boîte de dialogue. Vous pouvez également saisir, en lieu et place du nom, la valeur numérique d'un message (404h, par exemple). Ceci est utile si vous voulez interrompre la réception d'un message défini par l'utilisateur.

Certains messages Windows ont la même valeur. Ce n'est pas surprenant parce que ces messages iront toujours à des procédures de fenêtre différentes. Par exemple, TB_CHECKBUTTON, SB_GETTEXTA, TBM_GETRANGEMAX, PBM_SETPOS, HKM_GETHOTKEY, WM_PSD_MINMARGINRECT et DM_REPOSITION ont tous la valeur 402h. Notez que tous ces messages sont utilisés dans des procédures de fenêtres spécifiques et il n'y a donc aucun risque qu'ils soient confondus par le système. Comme ils ont la même valeur, GoBug n'opère un break sur cette valeur que si le message a une destination, ou dans le cas d'un message de notification, à une fenêtre du type approprié correspondant au message. La fenêtre Fantaisie visualise le type de fenêtre approprié au message.

Certains des messages répertoriés dans l'arborescence des messages sont envoyés sous forme de code avec l'un des supports de message de notification tels que WM_COMMAND, WM_NOTIFY, WM_DEVICECHANGE ou WM_POWERBROADCAST. Le code qui est envoyé est lui-même connu comme un "message de notification". Quand ceux-ci sont choisis pour le point d'arrêt, GoBug cherche le porteur de message et vérifie ensuite si le code envoyé correspond au message de notification choisi. GoBug vérifie également le type de fenêtre prévue pour envoyer le message. Tous les messages de notification sont répertoriés dans l'arborescence des messages. L'arborescence visualise le porteur de message entre parenthèses. Dans la tradition de Microsoft, tous les messages de notification négatifs sont affichés en valeurs décimales et non en hexadécimal comme d'habitude.

Comme de nouveaux messages sont ajoutés à chaque nouvelle version de Windows, ceux-ci ne seront probablement pas dans votre version de GoBug. Si vous trouvez qu'ils ne sont pas dans la dernière version de GoBug, ils peuvent être dans une mise à jour GoBug. S'il manque encore des messages, veuillez en informer Jeremy Gordon (JG@JGnet.co.uk). Pour le moment, lorsque vous entrez un point d'arrêt, vous pouvez entrer la valeur réelle du message dans le contrôle d'édition en bas de la boîte de dialogue.

Si des symboles sont chargés, vous pouvez limiter le point d'arrêt au message choisi ou aux messages atteignant une procédure de fenêtre particulière en cliquant sur la liste déroulante dans la partie supérieure de la boîte de dialogue. Initialement, cette zone de liste déroulante contient le nom de toutes les procédures se terminant par "proc" (non sensible à la casse). Il est classique d'appeler une procédure de fenêtre par un nom contenant proc. Mais vous pouvez obtenir les noms de toutes les autres procédures de code en cliquant sur le bouton "browse" ("Parcourir") en haut à droite de la boîte de dialogue. Revenez aux procédures se terminant par "proc" en cliquant à nouveau ici. Notez que, contrairement à d'autres boîtes de dialogue affichant des symboles, les symboles affichés dans la liste déroulante proviennent de tous les exécutables chargés en une seule fois (y compris les fichiers DLL) dans l'ordre alphabétique.

Lorsque vous utilisez ce point d'arrêt, le flag de trap est libéré et aucun journal d'instructions ou de messages n'est conservé tant que le point d'arrêt n'est pas atteint.

Si le message attendu n'arrive pas à l'endroit prévu, GoBug ne s'interrompt pas. Il pourrait y avoir plusieurs raisons pour que le message n'arrive pas. Il peut ne pas être distribué par la boucle de message ou être avalé par le système. Il existe des différences de comportement entre les différentes versions de Windows. En particulier, certains messages ne sont pas utilisés du tout dans certaines versions de Windows – voir le SDK.

3.3.8.8 Break en pas-à-pas

Les messages envoyés aux procédures de fenêtre de votre application provoquent également un break si vous travaillez en pas-à-pas (action F5) et que l'élément de menu "action, single-step message break" ("action, saut de

message à une seule étape") est coché. Pour plus d'informations à ce sujet, reportez-vous à la section Break de message en pas-à-pas.

3.3.8.9 Accès à l'adresse de code

Ce point d'arrêt vous permet de spécifier une adresse de code à laquelle l'exécution va s'arrêter et à partir de laquelle vous pouvez progresser en pas-à-pas. Afin de trouver où vous voulez votre point d'arrêt, vous aurez probablement besoin de visualiser la zone de code concernée dans le volet de code ou dans un inspecteur de code. En variante, l'adresse de code d'une procédure peut apparaître dans un fichier map produit par l'éditeur de liens.

Lorsque vous utilisez ce point d'arrêt, le flag de trap est libéré et aucun journal d'instructions ou de messages n'est conservé tant que le point d'arrêt n'est pas atteint.

Si des symboles sont chargés et reconnus par GoBug, vous trouverez une procédure plus facile à utiliser dans le paragraphe Accès à une procédure.

3.3.8.10 Accès à une procédure

Ce point d'arrêt vous permet de spécifier, en référence aux symboles de code, une adresse de code à laquelle l'exécution va s'arrêter et à partir de laquelle vous pouvez effectuer du pas-à-pas. Vous ne pouvez utiliser cette méthode de choix de point d'arrêt que si des symboles ont été chargés et reconnus comme tels par GoBug.

3.3.8.11 Accès à une exception ou une erreur d'API

GoBug effectue toujours un break si une exception se produit. GoBug effectuera également un break si une erreur d'API se produit si l'élément de menu "action, break on API error" ("action, pause sur l'erreur d'API") est cochée. Si vous souhaitez exécuter une erreur d'exception ou d'API, vous pouvez choisir d'utiliser :

F7	run/trap/hook/full log	exécution/hook/crochet/journal complet
F8	run/hook/part log	exécution/hook/journal partiel
F9	run in the background	exécution en arrière-plan

Le mode d'action que vous choisissez dépend de la quantité d'informations dont vous avez besoin lorsque l'exception ou l'erreur se produisent. Voir à ce sujet les paragraphes comment l'action de débogage affecte ce qui est consigné, ainsi que traçage de la pile.

3.3.8.12 Apparition des points d'arrêt précédents dans le menu

Pour vous aider à définir rapidement un point d'arrêt, vos dix précédents points d'arrêt pour le même debuggee apparaissent dans le menu "action". Cliquez sur l'un d'entre eux pour aller vers ce point d'arrêt. Si vous avez réassemblé ou recompilé votre programme depuis la dernière fois que vous avez utilisé l'un de ces points d'arrêt, vous pouvez constater que des adresses de code spécifiques ont été déplacées. Traitez-les avec précaution. Dans le cas des adresses de code choisies par un symbole, GoBug recherche à nouveau le même symbole, de sorte que l'adresse de code réelle d'un symbole particulier n'aura plus d'importance. Dans le cas de points d'arrêt de message, le menu affiche également le type de fenêtre et la procédure de fenêtre choisie si cela a été spécifié.

3.3.8.13 Abandon de l'accès à un point d'arrêt d'exécution

Cliquez sur "annuler" sur la fenêtre fantaisie qui apparaît au-dessus des volets principaux de GoBug lors de l'exécution d'un point d'arrêt.

3.3.8.14 Insertion de votre propre point d'arrêt de code

Double-cliquez sur le volet de code à l'adresse où vous voulez insérer un point d'arrêt de code.

Sinon, si le volet de code a le focus clavier, appuyez sur Entrée pour changer le rectangle de focus en surbrillance puis appuyez à nouveau sur Entrée pour constituer le point d'arrêt du code. Si la ligne est déjà en surbrillance, appuyez une seule fois sur Entrée.

En effet, cette manœuvre insère une INT 3 dans le code qui arrêtera l'exécution à ce point et amènera le debuggee dans la boucle de débogage. Vous pouvez définir autant de points d'arrêt de code que vous le souhaitez. Une ligne marron apparaît dans le volet de code de chacun. Si vous voulez définir un point d'arrêt de code en dehors de la section actuellement affichée dans le volet de code, cliquez sur l'élément de menu "inspect, codepane to view code at .., code address" ("inspecter, volet de code pour voir le code à .., adresse de code") pour accéder à une boîte de dialogue qui sera capable de visualiser d'autres sections et trouvez le bon endroit pour votre point d'arrêt. La définition d'un point d'arrêt de code a un effet *statique*, c'est-à-dire que vous pouvez choisir vous-même l'action de débogage qui provoquera l'exécution pour atteindre le point d'arrêt. Une fois là-bas, le debuggee sera dans la boucle de débogage et vous pourrez choisir n'importe quelle action – qui sautera par-dessus l'INT 3 dans le code, exécutera les opcodes originaux qui étaient là avant que l'INT 3 ne soit insérée, et continuera normalement.

3.3.8.15 Annulation des points d'arrêt de code

Vous pouvez annuler un point d'arrêt de code unique en double-cliquant dessus.

Sinon, si le volet de code a le focus, déplacez le rectangle de focus vers le point d'arrêt de code et appuyez sur Entrée. La ligne de point d'arrêt s'assombrit (indiquant qu'il est normalement en surbrillance), puis appuyez de nouveau sur Entrée. Si le point d'arrêt est déjà sombre, il vous suffit d'appuyer une fois sur Entrée.

Vous pouvez annuler tous les points d'arrêt de code à la fois en cliquant sur l'élément de menu "action, delete all code breakpoints" ("action, supprimer tous les points de rupture de code"). La définition d'un « Run to .. point d'arrêt » annulera également tous les points d'arrêt du code (à l'exception de Run to .. RET).

3.3.8.16 Insertion de votre propre point d'arrêt de code en utilisant l'INT 3

Si vous insérez le mnémonique INT 3 dans votre code source, l'exécution s'arrêtera à ce point et le debuggee entrera dans la boucle de débogage, vous permettant de faire du pas-à-pas à partir de ce point. C'est une méthode utile d'insertion de points d'arrêt si vous n'avez pas de symboles chargés que GoBug puisse reconnaître.

3.3.8.17 Insertion de votre propre point d'arrêt de code à l'aide de DebugBreak

Si votre assembleur ou compilateur ne vous permet pas d'insérer INT 3 dans votre code source, vous pouvez appeler l'API *DebugBreak*. Cela fait exactement la même chose, la seule différence étant que l'INT 3 est dans le code du système, puisqu'elle est appelée dans l'une des DLLs du système. Vous pouvez ensuite revenir facilement dans votre propre code source.

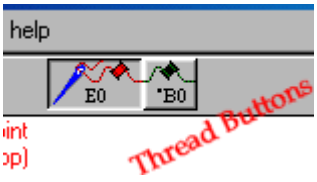
3.3.9 Applications Multi-thread

3.3.9.1 Applications Multi-thread

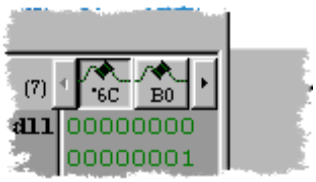
J'ai lu que les applications multithread devraient être évitées à tout prix. Certes, la vie serait alors plus facile. Mais la vérité est que l'utilisation des capacités multi-thread de Windows apporte quelques applications à la vie. GoBug est conçu spécifiquement pour gérer plus d'un thread dans votre application. Cependant, comme dans tout débogueur reposant sur le support de débogage Win32, il n'y a qu'une seule boucle de débogage. Et cela signifie que le débogage est fait sur une base de processus large. Si un thread est dans la boucle de débogage, l'ensemble du processus (y compris tous ses threads) est suspendu. Si vous faites ensuite un seul pas dans le code, le processus n'est pas suspendu et une instruction est exécutée. Cependant, le système peut décider de donner du temps-processeur à un autre thread. C'est peut-être cet autre thread qui vient ensuite dans la boucle de débogage. Cela permet de faire du pas-à-pas dans plus d'un thread, voir ci-après.

3.3.9.2 Les boutons de thread

Chaque thread dans le debuggee a un bouton de thread. Si un thread est actuellement dans la boucle de débogage, son bouton de thread sera plus large. Ici le thread E0h est à la porte d'entrée. Ce bouton de thread est coloré en rouge, indiquant que l'action du thread est en un seul pas. Le deuxième thread, B0h est l'action F9. Cela peut être vu parce que la couleur du thread est noire. B0h est aussi le thread principal. Ceci est indiqué par l'astérisque.



Le thread E0h est actuellement affiché dans les volets. Nous le savons parce que le bouton "thread" est "poussé". Le fait de cliquer sur le bouton de thread B0h affichera le dernier état connu du thread. Cependant, parce qu'il fonctionne (pas de trap), les résultats seront incertains. Par conséquent, les volets seront grisés si vous cliquez sur ce bouton.



Ici la fenêtre de GoBug est trop petite pour afficher tous les boutons de thread, mais vous pouvez les faire défiler tous dans la vue. Vous savez à partir du nombre entre parenthèses qu'il y a 7 boutons de thread en tout. Ici aucun des threads n'est à la porte d'entrée.

3.3.9.3 Contrôle de la manière dont le debuggee commence un nouveau thread

GoBug démarre de nouveaux threads avec l'action en cours, c'est-à-dire l'action effectuée en dernier par l'utilisateur. Ainsi, par exemple, si la dernière pression de touche sur GoBug était F9, tous les nouveaux threads auront la même action. Si vous voulez explorer en pas-à-pas un nouveau thread, utilisez simplement le menu "action" pour créer un point d'arrêt sur "nouveau thread". Si vous voulez vous assurer que vous attrapez un nouveau thread spécifique (au lieu de n'importe quel thread), insérez un point d'arrêt à l'adresse de code du début du thread (définissez un point d'arrêt sur le label d'adresse). Certains threads sont démarrés temporairement par le système. Par exemple, la boîte de dialogue système d'ouverture de fichier se trouvera dans un nouveau thread.

Ceux-ci ne sont pas interceptés par le nouveau point d'arrêt du thread de GoBug. De plus, ces threads démarrent toujours en arrière-plan (action F9).

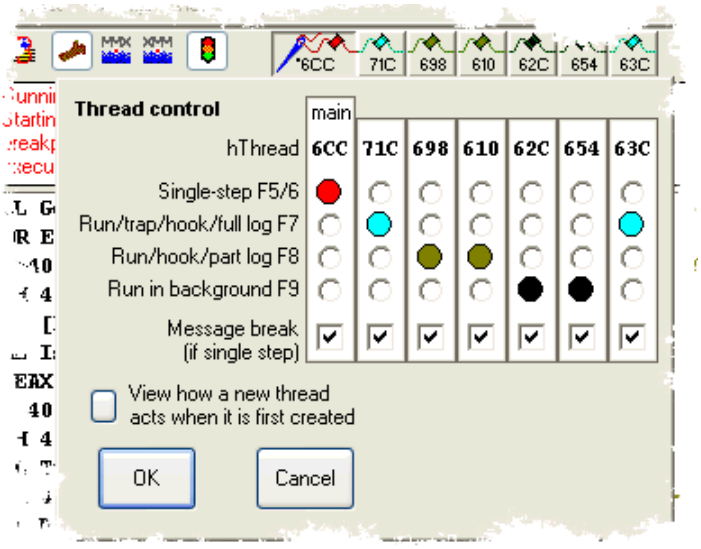
3.3.9.4 Pas-à-pas sur plus d'un thread

Le pas-à-pas sur plus d'un thread fournit un aperçu de la manière dont le système bascule le processeur entre les threads. Cela suppose que le système fonctionne de la même manière au ralenti. Quelqu'un dans la communauté Web peut-il confirmer cela ? Si c'est effectivement ainsi que le système fonctionne, vous verrez qu'il y a des commutations fréquentes entre les threads. Cela peut avoir des implications importantes au regard d'éventuels conflits de thread dans votre application multi-thread. En pratique lors du débogage, cela signifie que le thread dans la porte d'entrée changera souvent. Lorsque cela se produit, GoBug "balaye" les volets, c'est-à-dire que vous voyez les informations du volet correspondant au thread en cours de visualisation remplacées par celles du nouveau thread en cours de visualisation. Pour réaliser du pas-à-pas dans plus d'un thread, effectuez un seul pas via l'appel à *CreateThread* ou définissez un point d'arrêt à l'adresse de début des nouveaux threads ou utilisez le menu "action" pour créer un point d'arrêt sur "nouveau thread".

3.3.9.5 Concentration sur certains threads

Vous devrez peut-être faire du pas-à-pas sur un ou plusieurs threads particuliers sans avoir à passer par d'autres threads. Dans ce cas, exécutez les autres threads en utilisant F9. Pour ce faire, appuyez sur F9 chaque fois qu'un thread dans lequel vous n'avez aucun intérêt accède à la porte d'entrée. Sinon, vous pouvez attendre que tous les threads soient faits, puis utiliser le contrôle par feu tricolore pour provoquer l'exécution de tous les threads au regard desquels vous n'avez aucun intérêt. Actuellement, GoBug ne vous permet pas de suspendre un thread dans lequel vous n'avez aucun intérêt, bien que ce soit techniquement possible. Si quelqu'un aimerait que cette fonction soit ajoutée merci de me le faire savoir sur JG@JGnet.co.uk.

3.3.9.6 Contrôle par feux tricolores



Lorsque plusieurs threads sont en cours d'exécution, un clic sur le bouton de la barre d'outils du feu tricolore permet de visualiser et de contrôler le comportement des threads. Cela peut également être appelé depuis le menu "action" ou en utilisant la combinaison de touches Crtl+T.

Ici, nous voyons que le thread principal 6CCh (marqué d'un astérisque) est à la porte d'entrée et son action est en pas-à-pas. Les threads 71Ch et 63Ch sont définis sur F7, les threads 698h et 610h sur F8 et les threads 62Ch et 654h sur F9.

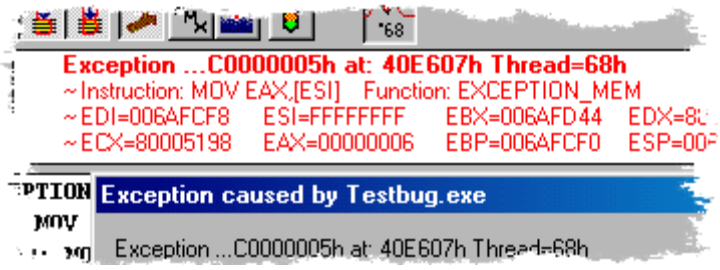
3.3.10 Débogage utilisant le journal

3.3.10.1 Le journal en tant qu'outil de débogage

Le journal est un outil précieux qui vous permet de poursuivre librement l'exécution du debuggee jusqu'à ce que quelque chose se passe mal, puis de voir ce qui s'est passé en consultant le journal. Lorsque vous utilisez GoBug de cette façon, vous pouvez trouver plus facile de mettre Gobug au premier plan. Cela permettra un accès plus facile au journal lorsque vous souhaitez l'afficher. Si vous utilisez le journal séquentiel, vous pouvez également le faire au premier plan. Lire, sur ce point, Plus sur GoBug au premier plan. Il existe quelques exemples d'utilisation du journal GoBug dans ce document.

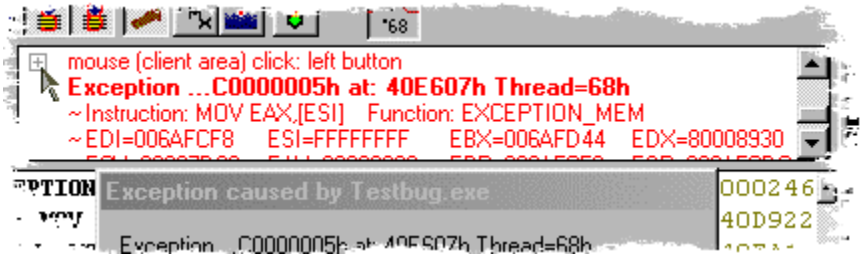
3.3.10.2 Poursuite jusqu'à l'exception, puis affichage du journal

Lorsqu'une exception se produit, GoBug est toujours interrompu et le journal affiche des informations sur l'exception:



Ici, la première ligne du journal fait apparaître le type d'exception, le contenu du pointeur d'instructions EIP où l'exception s'est produite et le thread. La deuxième ligne décrit l'instruction qui a provoqué l'exception et la fonction dans laquelle l'exception s'est produite (si elle est connue). Les valeurs des registres principaux *immédiatement avant l'exception* sont ensuite mentionnées.

Pour tirer le meilleur parti du journal, vous devez décider quelle action utiliser jusqu'à l'exception. Vous pouvez voir que les détails d'exception dans le journal sont un peu plus que ce qui est disponible de toute façon dans les volets lorsque GoBug fait un break sur une exception. C'est un *instantané* de la situation immédiatement avant l'exception. Cependant, si vous avez accédé seul à l'exception, ou si vous avez utilisé F7 pour exécuter et enregistrer ou Crtl-F5 pour l'animer, alors il y aura beaucoup plus d'informations disponibles dans le journal que dans les volets ordinaires quant aux causes de l'exception. C'est parce que le journal montre ensuite l'*historique* de l'exécution conduisant à l'exception. Cela peut être utile. Supposons par exemple qu'il existe une exception dans une procédure souvent appelée. Vous auriez probablement besoin de savoir ce qui a appelé la procédure afin d'identifier la cause de l'exception. Cela peut être vu à partir du journal, mais pas à partir des volets ordinaires.



Ici, l'utilisateur a fait défiler le journal d'une ligne et est prêt à voir ce qui a conduit à l'exception en cliquant sur la case "plus" juste avant l'exception.

Pour plus d'informations sur les exceptions, consultez la section Exceptions provoquées par le débogueur.

3.3.10.3 Animation sur un fragment de code puis affichage du journal

L'animation (en utilisant Crtl-F5) est un moyen pratique d'obtenir un enregistrement d'exécution sur une section de code suspect ou sur un code dont vous voulez vérifier l'exécution. Définissez un point d'arrêt au début du fragment de code et appuyez sur Crtl-F5. Observez le volet de code jusqu'à ce que l'exécution ait atteint la fin du fragment de code. Appuyez sur F5 pour arrêter l'animation. Voir le journal

3.3.10.4 Vérification des messages

La compréhension des messages et de la façon dont ils sont utilisés par le système sont au cœur de la programmation de Windows et vous pouvez suivre leur séquence dans le volet du journal des événements/messages ainsi que dans le journal séquentiel. Par exemple, vous pouvez vouloir ajuster la taille et la position d'une fenêtre subsidiaire lorsque son propriétaire est redimensionné par l'utilisateur mais vous n'avez pas encore décidé sur quel message vous allez le faire. L'action de déplacer ou de redimensionner une fenêtre génère une série de messages et vous pouvez les suivre dans le journal. La manière la plus simple est d'appuyer sur F8 lorsque le debuggee démarre. Cela permet au débogueur de s'exécuter, mais les hooks sont conservés afin que GoBug puisse enregistrer les messages. Voir l'affichage des messages dans le volet journal des événements/messages ou dans le volet journal séquentiel.

3.3.10.5 Éviter la duplication

Être capable de surveiller la séquence de messages à toutes vos fenêtres signifie que vous pouvez identifier ce qui relève de la *duplication*. Il arrive parfois, même dans le meilleur code, que le même message puisse être envoyé par le système de manière inattendue et traité par votre procédure de fenêtre une deuxième fois lorsque ce n'est pas nécessaire. Cela peut se produire par exemple si vous codez le rafraîchissement d'une fenêtre en cas d'événement alors, qu'en fait, le système le fait automatiquement.

3.3.10.6 Vérification de la création et de la terminaison de thread

Le journal s'affiche chaque fois qu'un thread est créé et à chaque fois qu'il est terminé (le code de terminaison est également affiché). Dans n'importe quel programme multithread, c'est une bonne idée de vérifier que cela se passe correctement en toutes circonstances, puisque vous ne voulez pas que votre programme continue les threads qui ne sont plus utilisés !

3.3.10.7 Visualisation de l'interaction du thread

Le journal vous permet de voir comment le système alloue les messages et l'heure du processeur aux threads individuels de votre programme. Vous pouvez regarder ceci quand vous êtes en pas-à-pas, mais en utilisant l'action F7 (run/hook/trap/full log), vous pouvez voir ceci dans le journal dans des circonstances plus réalistes. Comprendre cela vous aidera à vous assurer que vous n'êtes pas induit en erreur par les idées préconçues sur la synchronisation des threads, et servira de rappel sur le fait que chaque thread de votre programme doit être capable d'agir indépendamment.

3.3.10.8 Vérification du chargement et du déchargement des DLLs

Quand une DLL (ou un EXE d'ailleurs) est chargée, ou déchargée, de l'espace d'adressage du debuggee, cela apparaît dans le journal. Notez que la DLL n'est pas réellement écrite dans une autre partie de la mémoire dans ce processus. Si elle est déjà utilisée par une autre application, elle est présente en mémoire. Dans ce cas, le système fournit simplement une adresse virtuelle dans l'espace d'adressage propre au debuggee à partir de laquelle la DLL peut être lue, et sa section de données écrite.

3.3.10.9 Vérification des erreurs d'API

Les détails des erreurs de l'API sont archivés dans le journal et affichés dans tous les volets de journaux. Pour plus d'informations sur les erreurs d'API, consultez le paragraphe Erreurs d'API.

3.3.10.10 Envoi d'une chaîne dans le journal

Vous pouvez faire en sorte que le débogueur appelle l'API *OutputDebugString* lors de la survenue d'un événement dans le debuggee. Cette chaîne apparaîtra dans le journal (dans tous les volets du journal). Ceci est utile si vous voulez surveiller quelque chose dans le debuggee mais que vous ne voulez pas matérialiser sur l'écran le déroulement de l'événement. Cela peut être, par exemple, parce que l'événement que vous surveillez implique une peinture d'écran qui sera interférée si vous utilisez une fenêtre de testeur. Pour plus d'informations sur les fenêtres de test, consultez Ajouter un testeur visible à votre code au moment de l'erreur.

3.3.11 GoBug "on top"

Le fait que GoBug soit "au sommet" signifie que la fenêtre GoBug apparaîtra toujours au dessus du debuggee. Cela sera utile si vous voulez surveiller de près le debuggee et en même temps regarder sa sortie d'écran. Il est également généralement plus facile de mettre GoBug « top » (au sommet) si vous envisagez d'afficher le journal pour voir ce qui a conduit à une certaine erreur. Voir à ce sujet la section Débogage utilisant le journal. Pour rendre le GoBug "top", cliquez sur l'élément de menu "view, GoBug always on top" ("vue, GoBug toujours en haut"). Pour annuler cette option, utilisez le même élément de menu et annulez la vérification du menu.

Le journal séquentiel peut également être rendu "au sommet". Cliquez sur l'élément de menu "view, sequential log (always on top)" ("vue, journal séquentiel (toujours en haut)").

3.4 Obtention d'informations sur le système et le debuggee

3.4.1 Information sur le debuggee

3.4.1.1 La boîte de dialogue d'information

Elle consiste en une boîte de dialogue non-modale qui apparaît lorsque vous cliquez sur l'élément de menu "inspect, information about the debuggee" ("inspecter, informations sur le debuggee "). C'est un "instantané" montrant les informations en cours au moment où elle est ouverte. Elle ne s'actualise sur aucune action du débogueur.

3.4.1.2 L'information affichée

- A propos du débogueur lui-même (c'est-à-dire l'EXE principal du débogueur), rassemblé lors de sa création par le débogueur.
- A propos des DLLs qui ont été chargées par le debuggee, et qui restent chargées au moment de l'ouverture du dialogue. Celles-ci sont énumérées dans l'ordre dans lequel elles ont été chargées. Les adresses données sont

dans le propre contexte de la mémoire du débogueur, sauf qu'une adresse au-dessus de 7FFFFFFFh est en mémoire partagée.

- Dans le fichier PE, y compris les sections. Les sections et leurs adresses sont celles de l'EXE principal du debuggee *tel qu'il a été chargé*. Celles-ci différeront des adresses des différents éléments dans le fichier PE sur le disque. La signification des flags que vous rencontrerez habituellement (utilisés pour les caractères de section) est la suivante :

```
20h = Code
40h = Données initialisées
80h = Données non initialisées
200h = Informations ou commentaires
10000000h = Données partageables entre tous les processus utilisant la DLL
20000000h = Exécutable
40000000h = Lecture autorisée
80000000h = Écriture autorisée
```

- Zones de mémoire spécifiques au process. Celles-ci sont glanées du système et s'appliquent au moment où la boîte de dialogue est ouverte. Elles montrent diverses adresses dont certaines sont heureusement fournies par le système, d'autres ont dû être trouvées après quelques explorations spéléo.
- Zones de mémoire spécifiques au thread. Il s'agit d'informations spécifiques au thread pour chacun des threads du débogueur en cours d'exécution lors de l'ouverture de la boîte de dialogue.

3.4.1.3 Défilement de l'information

À l'aide de la souris, vous pouvez faire défiler l'information en cliquant sur les flèches de la barre de défilement ou sur la barre de défilement elle-même ou en faisant glisser le curseur de défilement.

L'interface du clavier (lorsque le dialogue d'information a le focus) est la suivante :

- UpArrow et DnArrow – monter ou descendre d'une ligne.
- PgUp et PgDn – monter ou descendre d'une page.
- Home et End – allez au début ou à la fin de l'information.

3.4.1.4 Vidage de l'information du dialogue

Assurez-vous que la boîte de dialogue d'information est visible et allez dans l'élément de menu "file, dump" ("fichier, vidage"). Cliquez sur le nom de la boîte de dialogue appropriée qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom). Choisissez un chemin d'accès pour que le fichier soit écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains dans vos noms de fichiers ou de sections). La boîte de dialogue d'information est toujours sauvegardée dans son intégralité. Le fichier créé sur cette image est du texte brut avec des retour-chariot et des sauts de ligne mais pas de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur paramétré sur une police de largeur fixe (par opposition à une police proportionnelle).

3.4.1.5 Impression de l'information du dialogue

Assurez-vous que la boîte de dialogue d'information est visible et passez à l'option de menu "file, print" ("Fichier, Imprimer"). Cliquez sur le nom de la boîte de dialogue appropriée qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). La boîte de dialogue d'information est toujours imprimée en entier (le nombre de pages apparaît dans la boîte de dialogue d'impression). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.4.2 Recherche et promenade dans la mémoire

3.4.2.1 Cartographie de la mémoire du debuggee (Windows 9x et ME)

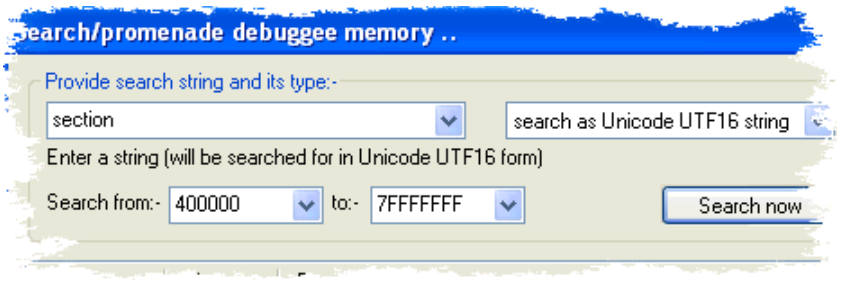
Zone de mémoire partagée C0000000h à FFFFFFFFh (3GB à 4GB)	Tables de pages mémoire et répertoire de pages composants Ring 0, handles, objets d'événement, contextes de mémoire, etc.
Zone de mémoire partagée 80000000h à BFFFFFFFh (2GB à 3GB)	Dll système et leurs heap Pseudo headers du PE Fichiers mappés en mémoire Bases de données de processus et de threads, bloc d'informations de thread Modules NE pour les exécutables 16 bits et 32 bits Files d'attente de messages, listes de fenêtres, etc.
Espace d'adressage propre du Debuggee 400000h à 7FFFFFFFh (4MB à 2GB)	Les heaps de mémoire alloués au Debuggee La pile du Debuggee L'environnement du Debuggee Le heap par défaut du Debuggee DLL du Debugee L'image propre du Debuggee
Zone de mémoire partagée 0h à 400000h (0 à 4MB)	Certains modules NE pour les exécutables 16 bits Certains DGROUPs 16 bits MS-DOS

3.4.2.2 Cartographie de la mémoire du debuggee (Windows NT, 2000 and XP)

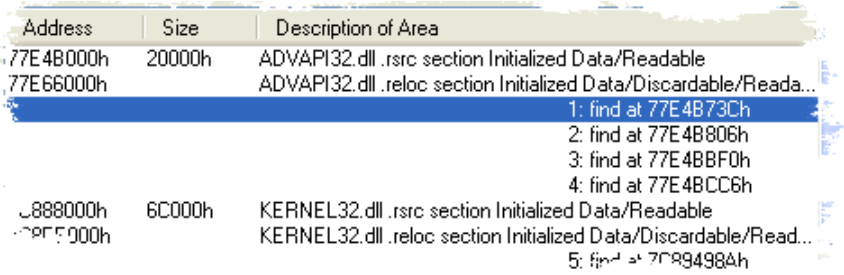
Zone de mémoire partagée C0800000h à FFFFFFFFh (à 4GB)	Cache système, pool paginé, code et données BIOS ROM Données utilisateur partagées Zone de contrôle du processeur, contexte de registre
Zone de mémoire système 80000000h à C0800000h	Table de vecteurs d'interruption, tables de descripteurs, BIOS VGA en ROM, services Kernel, répertoires de pages et tables
Bloc de garde supérieur 7FFEFFFFh to 7FFFFFFFh 64K juste en dessous de 2GB	Région inaccessible aux programmes destinée à intercepter les pointeurs sauvages causés par des erreurs de programmation courantes.
Espace d'adressage propre du Debuggee 10000h à 7FFEFFFFh (64K à 2GB *moins* 64K)	Heaps de mémoire alloués au Debuggee Pile du Debuggee Environnement du Debuggee heap par défaut du Debuggee Image propre du Debuggee Les DLLs de Debuggee DLLs système utilisées par le debuggee
Bloc de garde inférieur 0h à 10000h (0 à 64K)	Région inaccessible aux programmes destinée à intercepter les pointeurs sauvages causés par des erreurs de programmation courantes.

3.4.2.3 Recherche en mémoire

L'élément de menu "Inspect, search/promenade memory" ("Inspecter, rechercher / lancer la mémoire") produit la boîte de dialogue search/promenade (recherche/navigation). Vous pouvez l'utiliser pour rechercher dans l'espace d'adressage du debuggee (et dans la mémoire partagée) des chaînes spécifiques (nombres ou texte) qui, selon vous, pourraient être en mémoire quelque part.



Ici, l'utilisateur a changé la combo box sur le côté droit pour "search as Unicode UTF16 string" ("rechercher en tant que chaîne UTF16 Unicode") et a entré la chaîne "section". En cliquant sur "Search now" ("rechercher maintenant") cela donne lieu à un certain nombre de "trouvailles" :



Notez que si vous utilisez GoBug sous Windows NT/2000/XP, la plupart des chaînes seront conservées par le système au format Unicode. Le format UTF16 représente la plupart des caractères en tant que mots de 16 bits.

Si vous exécutez GoBug sous Windows 9x ou ME, la plupart des chaînes seront conservées par le système au format ANSI.

Vous pouvez choisir le format à utiliser pour la recherche en déroulant la combo box. Vous pouvez choisir ainsi les formats dwords, word ou bytes (uniquement pour les nombres hexadécimaux), Unicode UTF16, Unicode UTF8 ou ANSI.

Dans certains cas, vous êtes en mesure de trouver la chaîne que vous recherchez réellement, détenue par le système lui-même.

Souvenez-vous que cette recherche porte sur l'espace d'adresses et les zones de mémoire partagée du *debuggee*.

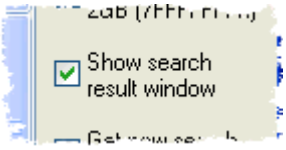
3.4.2.4 Spécification des zones de recherche

Au lieu de spécifier directement la zone de recherche dans les combo box "Search from" et "to:", vous pouvez spécifier la zone plus rapidement en cochant la case "Get new search area from selection" ("Obtenir une nouvelle zone de recherche à partir de la sélection"). Vous pouvez ensuite utiliser le contenu de la zone de résultat pour alimenter la zone de recherche. Utilisez les touches Crtl et Shft avec la souris pour sélectionner la zone que vous souhaitez rechercher.

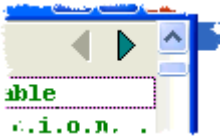
Sous Windows 9x et ME, la recherche a tendance à être lente au-dessus de 3 Go. C'est parce que l'API *VirtualQueryEx* semble fonctionner très lentement au-dessus de cette adresse. Par conséquent, vous pouvez activer ou désactiver la recherche au-dessus de cette adresse.

Dans Windows NT, 2000 et XP normalement, vous ne pouvez pas rechercher au-delà de 2 Go.

3.4.2.5 Affichage des résultats de la recherche



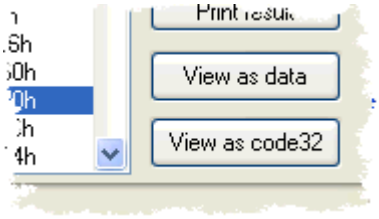
[1]



[2]

Il y a deux façons de visualiser la zone de mémoire qui fait l'objet d'une "recherche". La plus simple est de cocher la case "Show search result window" ("Afficher la fenêtre de résultats de recherche") [1]. Cela provoque l'apparition d'un inspecteur spécial nommé "Search inspector" ("Inspecteur de recherche") lorsqu'une la recherche aboutit. Cet inspecteur possède des boutons de défilement spéciaux gauche/droite qui vous permettent de passer rapidement d'un résultat de recherche au suivant [2].

L'autre façon de visualiser la zone de mémoire qui fait l'objet d'une



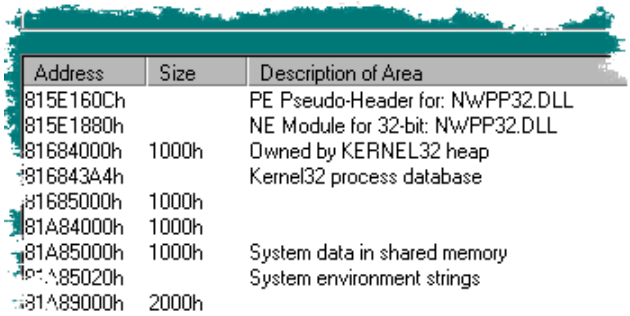
[3]

"recherche" est de mettre en surbrillance la recherche dans la boîte de résultats [3], puis de cliquer sur les boutons "View as data" ("Afficher en tant que données") ou "View as code 32" ("Afficher en tant que code 32 bits"). Utilisez les touches Crtl ou Shft pour sélectionner plusieurs zones à afficher. Les boutons "Vue" créent des inspecteurs de données ou de codes qui disparaissent normalement lorsque la boîte de dialogue est fermée. Si vous voulez qu'ils restent en tant que panneaux d'inspecteur, cliquez sur "garder les inspecteurs"

3.4.2.6 Promenade en mémoire

L'élément de menu "Inspect, search/promenade memory" ("Inspecter, rechercher / se promener en mémoire") produit la boîte de dialogue de "search/promenade" ("recherche/navigation"). Si vous cliquez sur le bouton "promenade", GoBug teste chaque zone de mémoire à partir de zéro. GoBug vérifie s'il est autorisé à lire la zone de mémoire et si c'est le cas, il signale la zone dans la zone de résultat.

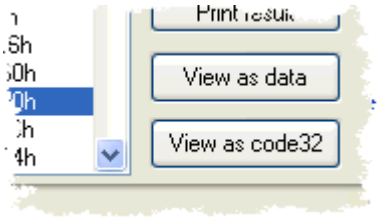
Les résultats que vous voyez dépendent de la version de Windows sur laquelle vous exécutez GoBug. Sous 9x et ME, l'information tend à être plus étendue, car il y a plus de zones internes dont l'observation est permise. Voici quelques résultats typiques sous Windows 98 :



Address	Size	Description of Area
815E160Ch		PE Pseudo-Header for: NWPP32.DLL
815E1880h		NE Module for 32-bit: NWPP32.DLL
81684000h	1000h	Owned by KERNEL32 heap
816843A4h		Kernel32 process database
81685000h	1000h	
81A84000h	1000h	
81A85000h	1000h	System data in shared memory
81A85020h		System environment strings
81A89000h	2000h	

Si GoBug reconnaît la zone de mémoire ou son contenu, il donnera plus d'informations à ce sujet, sinon il n'y aura rien sous "Description of Area" ("Description de la zone"). GoBug utilise l'API *VirtualQueryEx* pour vérifier chaque zone de mémoire, et les colonnes "address" et "size" proviennent des résultats de l'appel. Sous Windows 9x et ME au-dessus de 3 Go, cette API fonctionne très lentement, ce qui explique pourquoi vous pouvez activer ou désactiver la promenade au-dessus de cette adresse. Dans Windows NT, 2000 et XP, normalement, vous ne pouvez pas lire la mémoire au-dessus de 2 Go de sorte que le commutateur est défini à cette valeur lors de l'exécution de GoBug sur ces systèmes.

3.4.2.7 Affichage des résultats de la promenade



Pour visualiser une zone de mémoire affichée par la promenade, cliquez sur la zone de résultat et cliquez sur "View as data" ("Afficher comme donnée") ou sur "View as code 32" ("Afficher comme code 32"). Utilisez les touches Crtl ou Shft pour sélectionner plusieurs zones à afficher. Les boutons "View" créent des inspecteurs de données ou de codes qui disparaissent normalement lorsque la boîte de dialogue est fermée. Si vous voulez qu'ils restent en tant que volet d'inspecteur, cliquez sur "keep any inspectors made" ("garder les inspecteurs constitués"). Alors que la boîte de dialogue de la promenade est ouverte, les inspecteurs de données en savent un peu plus sur les zones de mémoire que d'habitude. En effet, la promenade prend un instantané des zones de mémoire à l'aide de l'API *Toolhelp*. C'est une information temporaire qui est perdue quand la boîte de dialogue de la promenade est fermée.

3.4.2.8 Zones exclues

Sous Windows 9x et ME, il n'y a pas de promenade ou de recherche entre 0A0000h et 0AFFFFh. Ceci afin d'éviter certains problèmes qui ont été trouvés lorsque GoBug était en test en essayant de lire dans cette zone de mémoire.

Voici une explication pour certaines zones de mémoire pas si évidentes que vous verrez dans les résultats de la promenade lorsque vous travaillerez sous 9x et ME :

Un **pseudo-header PE** est une base de données contenant des informations sur un exécutable 32 bits. Le système sous 9x et ME crée cette base de données pour chaque exécutable 32 bits chargé au format IMAGE_FILE_HEADER (défini dans le fichier d'en-tête Microsoft Winnt.h). La base de données est utilisée par le système pour trouver rapidement les composants de l'exécutable. Ces zones n'existent pas sous NT/2000/XP.

Un **module NE** est une base de données contenant des informations sur l'exécutable à utiliser par le système. Il commence toujours par la signature "NE" pour "New Executable". Le système crée des modules NE pour les exécutables 32 bits et 16 bits sous 9x et ME, mais pas sous NT/2000/XP.

3.4.2.9 Vidage des résultats de la recherche/promenade

Cliquez sur le bouton "dump results" ("Vider les résultats") dans la boîte de dialogue search/promenade (recherche/promenade). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom). Choisissez un chemin d'accès pour que le fichier soit écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains dans vos noms de fichiers et de sections). Les résultats sont toujours sauvegardés dans leur intégralité. Le fichier créé issu de ce vidage est du texte brut avec des retours chariot et des sauts de ligne mais exempt de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur défini sur une police de largeur fixe (par opposition à une police proportionnelle). $$$

3.4.2.10 Impression des résultats de la recherche/promenade

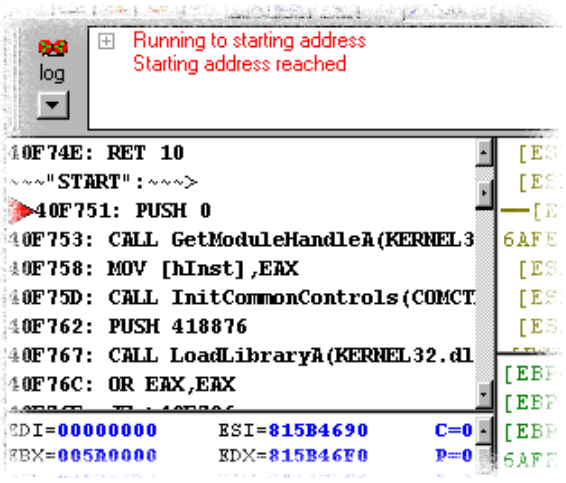
Cliquez sur le bouton "print results" ("imprimer les résultats") dans le dialogue search/promenade (recherche/promenade). Les résultats sont toujours imprimés en entier (le nombre de pages apparaît dans la boîte de dialogue d'impression). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.4.3 Symboles de Code et de Données

3.4.3.1 Affichage des symboles de code

L'élément de menu "inspect, EXE/DLL code symbols" produit une boîte de dialogue non modale dans laquelle vous pouvez afficher les symboles de code du debuggee et des DLL chargées, à condition que des symboles soient disponibles. L'adresse du symbole de code est également donnée. C'est l'adresse dans le propre espace d'adressage du debuggee. Le nom de la section est également donné. Vous pouvez regarder aussi une liste de symboles de code dans toutes les DLL si elles sont disponibles. Utilisez la liste déroulante pour changer l'EXE/DLL en cours de visionnage. Vous pouvez créer un inspecteur de code en cliquant sur le bouton "View as code" ("Afficher en tant que code") dans cette boîte de dialogue.

3.4.3.2 Le symbole de code "START" ou "DLLSTART"



Si GoBug ne connaît pas le symbole à l'adresse de départ de l'exécutable, il utilise en lieu et place ces symboles simulés. Ceux-ci apparaîtront dans le volet de code ou l'inspecteur de code. Ils apparaîtront également dans les listes de symboles. Les symboles simulés apparaissent toujours entre guillemets. En fait, vous aurez probablement donné à votre programme un symbole à l'adresse de départ que vous aurez passée à l'éditeur de liens. Cependant, étant donné que l'adresse de début de l'exécutable est détenue dans les parties formelles du fichier PE lui-même, l'éditeur de liens aura généralement ignoré le nom du symbole lui-même. Cela explique pourquoi GoBug peut ne pas connaître le symbole à l'adresse de départ et pourquoi il est nécessaire de le simuler.

3.4.3.3 Affichage des symboles de données

L'élément de menu "inspect, exe/dll data symbols" ("inspecter, symboles de données EXE/DLL") génère une boîte de dialogue non modale dans laquelle vous pouvez afficher les symboles de données du debuggee et des DLL chargées à condition que des symboles soient disponibles. L'adresse du symbole de données est également donnée. C'est l'adresse dans le propre espace d'adressage du debuggee. Le nom de la section est également donné. Vous pouvez également regarder une liste de symboles de données dans n'importe quelle DLL s'ils sont disponibles. Utilisez la liste déroulante de la combo box pour changer l'EXE/DLL en cours de visionnage. Vous pouvez créer un inspecteur de données en cliquant sur le bouton "View as data" ("Afficher en tant que données") dans cette boîte de dialogue.

3.4.3.4 Vidage des symboles de code et de données

Assurez-vous que la boîte de dialogue de symboles de code ou de données (selon le cas) est visible et accédez à l'option de menu "file, dump" ("fichier, vidage"). Cliquez sur le nom de la boîte de dialogue appropriée qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom à cet effet). Choisissez un chemin d'accès pour que le fichier soit écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un

fichier Unicode car vous utilisez des caractères non-romains dans votre code ou vos symboles de données). Les symboles de code ou de données sont toujours sauvegardés intégralement. Le fichier issu de ce vidage est du texte brut avec des retours chariot et des sauts de ligne tout en étant exempt de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur paramétré avec une police de largeur fixe (par opposition à une police proportionnelle).

3.4.3.5 Impression des symboles de code et de données

Assurez-vous que la boîte de dialogue de symboles de code ou de données (selon le cas) est visible et accédez à l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur le nom de la boîte de dialogue approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Les symboles de code ou de données sont toujours imprimés en entier (le nombre de pages apparaît dans la boîte de dialogue d'impression). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.4.4 Le Debuggee et les adresses de DLLs

3.4.4.1 Boîte de dialogue des adresses EXE/DLL

L'élément de menu "inspect, exe/dll addresses" génère une boîte de dialogue non modale dans laquelle vous pouvez afficher les différentes parties du debuggee et des DLL chargées. Ce sont les parties formelles et les différentes sections. Les tailles de section montrées ici incluent le comblement ; il est peu probable que toute la zone contienne des données. Vous pouvez créer un inspecteur de code en cliquant sur le bouton View as code" ("Afficher en tant que code") dans cette boîte de dialogue ou sur un inspecteur de données en cliquant sur le bouton "View as data" ("Afficher en tant que données").

3.4.4.2 Changement de l'EXE/DLL en cours d'examen

Utilisez la liste déroulante de la combo box pour faire cela. Les fichiers présentés ici sont les fichiers actuellement chargés par le debuggee dans son propre espace d'adressage ou dans la mémoire partagée.

3.4.4.3 Vidage des adresses EXE/DLL

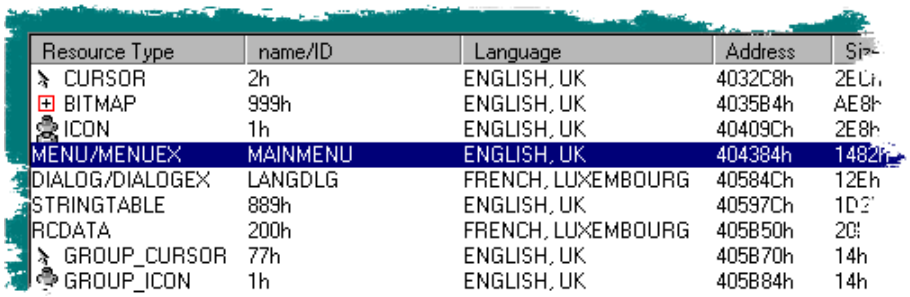
Assurez-vous que la boîte de dialogue d'adresses EXE/DLL est visible et allez à l'option de menu "file, dump" ("fichier, vidage"). Cliquez sur le nom approprié de la boîte de dialogue qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom). Choisissez un chemin d'accès pour que le fichier puisse être écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains dans vos noms de fichiers ou de sections). Les adresses EXE/DLL sont toujours sauvegardées dans leur intégralité. Le fichier issu de ce vidage est du texte brut avec des retours chariot et des sauts de ligne mais exempt de tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur paramétré avec une police de largeur fixe (par opposition à une police proportionnelle).

3.4.4.4 Impression des adresses EXE/DLL

Assurez-vous que la boîte de dialogue de l'adresse EXE/DLL est visible et allez dans l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur le nom de la boîte de dialogue approprié qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). Les adresses EXE/DLL sont toujours imprimées en entier (le nombre de pages apparaît dans la boîte de dialogue d'impression). Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.4.5 Le Debuggee et les ressources de DLL

3.4.5.1 Le dialogue de ressource

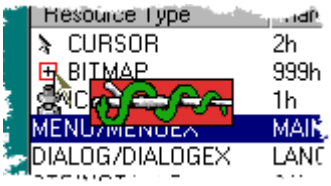


Resource Type	name/ID	Language	Address	Si
CURSOR	2h	ENGLISH, UK	4032C8h	2EC
BITMAP	999h	ENGLISH, UK	4035B4h	AE8
ICON	1h	ENGLISH, UK	40409Ch	2E8
MENU/MENUEX	MAINMENU	ENGLISH, UK	404384h	1482
DIALOG/DIALOGEX	LANGDLG	FRENCH, LUXEMBOURG	40584Ch	12Eh
STRINGTABLE	889h	ENGLISH, UK	40597Ch	1D2
RCDATA	200h	FRENCH, LUXEMBOURG	405B50h	20
GROUP_CURSOR	77h	ENGLISH, UK	405B70h	14h
GROUP_ICON	1h	ENGLISH, UK	405B84h	14h

L'élément de menu "inspect, EXE/DLL resources" ("inspection, ressources EXE/DLL ") produit une boîte de dialogue non modale dans laquelle vous pouvez afficher les ressources dans la section ressources

et peut comprendre des icônes, des bitmaps, des menus, des dialogues, des informations de version ainsi que d'autres données.

Notez qu'un curseur ou une icône possèdent toujours une entrée de ressource "curseur de groupe" ou "icône de groupe", ainsi qu'une entrée pour leurs propres données brutes. Les entrées de groupe contiennent des informations sur la taille et la couleur du curseur ou de l'icône utilisés par le système pour déterminer la version du curseur ou de l'icône à utiliser pour l'affichage. Dans les ressources présentées ici, il n'y a qu'une seule version du curseur ou de l'icône. L'ID du curseur ou de l'icône est simplement une numérotation à partir de 1 insérée par le compilateur de ressources.



L'adresse de chaque ressource correspond à celle du début de la ressource (c'est-à-dire, les données brutes de la ressource, pas son entrée dans le répertoire) dans le fichier EXE tel que chargé par le système. Les bitmaps apparaissent lorsque vous déplacez le curseur de la souris sur les petites boîtes d'expansion rouges.

3.4.5.2 Changement de l'EXE/DLL en cours d'examen

Utilisez la liste déroulante combo box pour faire cela. Les fichiers présentés ici sont les fichiers actuellement chargés par le debuggee dans son propre espace d'adressage ou dans la mémoire partagée.

3.4.5.3 Affichage d'une ressource

Pour visualiser une ressource, cliquez sur celle que vous voulez regarder et cliquez sur le bouton "View". Une fenêtre d'inspecteur apparaît (semblable à une fenêtre d'inspecteur de données) et vous montre les données dans la ressource que vous avez choisie. Le bouton "View All" ("Afficher tout") vous permet d'afficher les données de toutes les ressources dans un seul inspecteur. Notez que, si vous voulez visualiser la section entière de ressources (y compris tous les répertoires) vous devriez employer l'élément de menu "inspect, create new data inspector by .., exe/dll addresses" ("inspectez, créez le nouvel inspecteur de données par .., adresses d'exe / dll").

3.4.5.4 Extraction d'une ressource

Cette fonctionnalité vous permet d'enregistrer une ressource dans un fichier. Vous pouvez enregistrer une ressource dans un fichier ".res", dans un format adapté à l'édition de liens, ou dans un fichier de données, que vous pouvez analyser à l'aide d'un éditeur de texte ou d'un éditeur hexadécimal tel que Paws (voir http://www.GoDevTool.com), ou dans le cas d'un curseur, d'une icône ou d'une image bitmap vers un fichier ".cur", ".ico" ou ".bmp" respectivement. Notez que lorsque vous êtes invité à entrer un nom de fichier dans la boîte de dialogue Windows "Enregistrer", GoBug suggère un nom pour le fichier. C'est un nom généré au hasard par GoBug. Vous pouvez l'utiliser ou le changer par le nom que vous souhaitez.

Notez que les icônes et les curseurs ont toujours une «icône de groupe» ou un «curseur de groupe» correspondant. Cela permet de contenir différentes versions et tailles de l'icône ou du curseur dans l'icône de groupe ou le curseur de groupe. Vous pouvez extraire une icône individuelle ou extraire une icône de groupe ou un curseur pour extraire toutes les icônes ou tous les curseurs du groupe.

3.4.5.5 Utilisation du clavier dans la boîte de dialogue des ressources

L'interface du clavier dans la boîte de dialogue des ressources est la même que pour les autres dialogues sauf que vous pouvez afficher une ressource bitmap en sélectionnant la ligne contenant le nom du bitmap en utilisant les flèches haut et bas lorsque le focus est dans le contrôle Listview, puis actionnez la flèche droite pour faire apparaître l'image bitmap et la flèche gauche pour l'enlever. Utilisez les flèches haut et bas pour voir les autres bitmaps.

3.4.5.6 Vidage de la liste de ressources

Assurez-vous que la boîte de dialogue de ressources est visible et accédez à l'élément de menu "file, dump" ("fichier, vidage"). Cliquez sur le nom de la boîte de dialogue appropriée qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "dump the pane" ("vider le volet"). Choisissez un nom de fichier pour le fichier sauvegardé (GoBug proposera un nom). Choisissez un chemin d'accès pour que le fichier soit écrit (GoBug s'en souviendra pour la prochaine fois que vous déboguerez cet EXE particulier). Choisissez le type de fichier à créer (utilisez ANSI sauf si vous voulez créer un fichier Unicode car vous utilisez des caractères non-romains dans les noms ou types de ressources). La liste des ressources est toujours sauvegardée dans son intégralité. Le fichier issu de ce vidage est du texte brut avec des retours chariot et des sauts de ligne sans tabulations. Il est généralement préférable d'afficher le fichier sauvegardé avec un éditeur paramétré sur une police de largeur fixe (par opposition à une police proportionnelle).

3.4.5.7 Impression de la liste de ressources

Assurez-vous que la boîte de dialogue des ressources est visible et accédez à l'élément de menu "file, print" ("Fichier, Imprimer"). Cliquez sur le nom de la boîte de dialogue appropriée qui apparaîtra dans l'élément de menu. Sinon, faites un clic droit sur le volet et choisissez "print the pane" ("imprimer le volet"). La liste des ressources est toujours sauvegardée dans son intégralité. Vous pouvez modifier la police utilisée pour l'impression en utilisant l'élément de menu "settings, fonts, when printing" ("paramètres, polices, lors de l'impression").

3.5 Personnalisation de GoBug

3.5.1 Le journal

Cliquez sur l'élément de menu "settings, log settings" ("paramètres, paramètres de journal"). Vous pouvez maintenant modifier les messages ignorés dans le journal, la quantité de mémoire que le journal peut utiliser, ainsi que la police et la couleur du texte du journal.

Voir :

- Ignorer certains messages
- Couleurs du texte du journal et polices de caractères
- Mémoire utilisée par le journal

3.5.2 La touche de raccourci

Cliquez sur l'élément de menu "settings, define hot-key" ("paramètres, définir une touche de raccourci"). Appuyez sur les touches à utiliser pour constituer la nouvelle touche de raccourci. L'écran vous indiquera si vous pouvez ou non utiliser cette touche ou combinaison de touches. Si vous avez défini la touche à utiliser, cliquez sur "Appliquer". Il est préférable d'utiliser une combinaison de touches inhabituelle afin que la touche de raccourci ne puisse pas être invoquée en utilisation normale. Crtl et/ou Shift et une touche F est généralement une bonne solution. Évitez d'utiliser des combinaisons utilisées par Windows comme, par exemple, Alt+Tab, Alt+Shft+Tab, Alt+F4 ou Alt+Esc. Évitez de même les combinaisons utilisées comme raccourcis par l'application que vous avez l'intention de déboguer. Cliquez sur ce lien pour obtenir plus d'informations la touche de raccourci.

3.5.3 Couleurs

Cliquez sur l'élément de menu "settings, colours" ("paramètres, couleurs") et choisissez le volet dont vous voulez changer la couleur.

3.5.4 Police de caractères

Cliquez sur l'élément de menu "settings, fonts" ("paramètres, polices") et choisissez le volet dont vous voulez changer la police. Les volets ne changent pas de taille lorsque vous changez la police. Vous devrez les redimensionner à la mesure de votre goût. Les volets MMX et Floating Point sont différents. Ils se redimensionnent eux-mêmes afin de pouvoir afficher tout leur contenu dans la nouvelle police. L'autre différence est que vous êtes limité à l'emploi d'une police non proportionnelle avec ces volets.

3.5.5 Largeur des barres de défilement de volet

Si les barres de défilement étroites du volet (par défaut) ne vous conviennent pas, cliquez sur l'élément de menu "settings, pane scrollbar widths" ("paramètres, largeur de la barre de défilement du volet") pour les modifier.

3.5.6 Polices, tailles et couleurs du système

Les polices, tailles et couleurs du système peuvent être modifiées à l'aide du module Affichage, Apparences du Panneau de configuration de Windows. GoBug répond habituellement aux changements immédiatement, mais parfois seulement après que les fenêtres et des volets particuliers ont été reconstitués à l'occasion d'un changement quelconque. L'agrandissement de la police du système agrandira toutes les boîtes de dialogue de GoBug, mais les polices utilisées dans les fenêtres et les fenêtres autres que les boîtes de dialogue resteront inchangées. Changez-les en utilisant l'élément de menu de GoBug "settings, fonts" ("paramètres, polices"). Vous pouvez changer la couleur d'arrière-plan des fenêtres et des volets, mais la couleur du texte doit être modifiée en utilisant l'élément de menu de GoBug "settings, colors" ("paramètres, couleurs").

3.6 Messages d'erreur et dépannage

Ce paragraphe commente les principaux messages qui peuvent apparaître ou des phénomènes non évidents qui peuvent se produire lorsque vous utilisez GoBug. Tous les messages que vous pourriez rencontrer ne sont pas relatés ici. Les messages qui n'ont pas besoin d'une planification préalable ne sont pas traités.

3.6.1 Problèmes lors du démarrage de GoBug

"This executable requires a file which is missing - HHCTRL.OCX"

"Cet exécutable nécessite un fichier qui manque - HHCTRL.OCX"

Cause : GoBug s'appuie sur les fichiers système de Windows pour ses fonctionnalités, et il ne démarrera pas si l'un d'eux est manquant. S'il vous manque HHCTRL.OCX (un fichier système qui montre ce fichier d'aide) exécutez le fichier auto-extractible hhupd.exe (mise à jour de l'aide). Il est disponible sur le site Web de Microsoft.

"You need Version 2 to run under this operating system."

"Vous avez besoin de la version 2 pour fonctionner sous ce système d'exploitation."

Cause : Vous utilisez GoBug Version 1.xx sous Windows NT, 2000, XP ou Vista. Vous avez besoin de la version 2.xx Visitez www.GoProg.com.

"GoBug asked the system for debug privileges, but was not granted them ..."

"GoBug a demandé au système des privilèges de débogage, mais il ne leur a pas accordé ..."

Cause : Vous vous êtes connecté à Windows NT, 2000, XP ou Vista et votre nom d'utilisateur n'a pas de privilèges de débogage. Dans Windows NT, 2000 ou XP, connectez-vous en tant qu'administrateur ou demandez à l'administrateur d'accorder vos privilèges de débogage à votre nom d'utilisateur. Dans Vista ou version ultérieure, pour une sécurité accrue au cas où un logiciel malveillant aurait infecté la machine, l'administrateur n'a pas automatiquement de privilèges de débogage. Vous devrez probablement augmenter votre niveau de privilège pour pouvoir utiliser GoBug. Pour ce faire, faites un clic droit sur GoBug.exe, cliquez sur "propriétés", sélectionnez l'onglet "compatibilité", et sous "privilege level" ("niveau de privilège") sélectionnez "exécuter ce programme en tant qu'administrateur", puis cliquez sur OK. Vous pouvez également désactiver le contrôle de compte d'utilisateur ("UAC" ou User Account Control) dans le panneau de configuration, comptes d'utilisateurs.

"GoBug is already running. To run GoBug more than once at the same time, change its name, or run it in a different directory."

"GoBug est déjà en cours d'exécution Pour exécuter GoBug plus d'une fois en même temps, changez son nom ou exécutez-le dans un répertoire différent."

Cause : Une copie de GoBug est déjà en cours d'exécution sur la même machine. Vous ne pouvez pas exécuter deux instances de GoBug sur la même machine sans changer son nom, ou (mieux) sans l'exécuter dans un autre répertoire, voir GoBug.exe.

"The file containing the GoBug configuration (GoBug.ini) was found to be the wrong version or was corrupted. Do you agree that this may be deleted and replaced?"

"Le fichier contenant la configuration de GoBug (GoBug.ini) a été trouvé dans une mauvaise version ou a été corrompu." Êtes-vous d'accord pour qu'il soit supprimé et remplacé ? "

Cause : De temps à autre, le format de fichier *ini* peut changer dans les nouvelles versions de GoBug. Dans ce cas, si vous passez à la nouvelle version et que vous n'avez pas supprimé un fichier *ini* qui était utilisé par une version précédente, ce message apparaîtra.

"The file containing the GoBug configuration (GoBug.ini) was not found so a new one was made"

"Le fichier contenant la configuration GoBug (GoBug.ini) n'a pas été trouvé et un nouveau fichier a donc été créé"

Cause : Cela peut apparaître lorsque vous utilisez GoBug pour la première fois, car le fichier *ini* n'est normalement pas livré avec. Ce fichier doit être dans le même répertoire que GoBug. Si vous modifiez le nom de GoBug.exe pour pouvoir l'exécuter plus d'une fois, vous verrez ce message car un nouveau fichier *ini* est créé et correspond au nouveau nom de l'exécutable. Lorsqu'un nouveau fichier *ini* est créé, toutes les valeurs par défaut de GoBug sont restaurées.

"The system has reported that GoBug's current hot key is already reserved by another application. To use a hot key you will need to set a different one using the settings menu. Alternatively close the other application and restart GoBug, or if necessary, reboot."

"Le système a signalé que la touche de raccourci actuelle de GoBug est déjà réservée par une autre application. Pour utiliser une touche de raccourci, vous devrez en définir une différente en utilisant le menu des paramètres. Sinon, fermez l'autre application et redémarrez GoBug ou, si nécessaire, redémarrer le système. "

Causes :

1. Il existe une autre application qui utilise la même touche de raccourci que GoBug. La touche de raccourci par défaut de Gobug est Shift+Pause. Si une autre application a déjà réservé cette touche de raccourci, vous pouvez la modifier à partir du menu des paramètres de GoBug.

2. Vous avez deux copies ou plus de GoBug en cours d'exécution en même temps. Normalement, GoBug ne vous permettra pas de faire cela, mais cela peut être fait si vous changez le nom de l'EXE ou si vous lancez GoBug à partir d'un autre répertoire. Voir GoBug.exe.

3. GoBug veille, quand il se termine normalement ou via son gestionnaire d'exceptions, à effacer toutes les ressources qu'il a utilisées, y compris la touche de raccourci, qui sera ensuite effacée du système. En fait, même si un programme ne prend pas la peine d'effectuer cette opération, le système agira de même. Cependant, il est possible que ni GoBug ni le système n'aient pu effacer correctement, de sorte que la touche de raccourci reste enregistrée dans le système. Dans ce cas, lorsque GoBug redémarrera, vous verrez le message ci-dessus. Si cela se produit et si vous voulez utiliser la touche de raccourci, vous devrez en enregistrer une autre à partir du menu des paramètres de GoBug. Vous pouvez également redémarrer, ce qui effacera toutes les touches de raccourci du système.

>> I am starting GoBug normally but the main window does not appear.

Je démarre normalement GoBug mais la fenêtre principale n'apparaît pas.

Cause : La cause la plus probable de ce problème est que le fichier *GoBug.ini* est corrompu (on rappelle que GoBug reçoit ses instructions pour la taille et la position de la fenêtre principale du fichier *ini*).

Solution : Supprimez le fichier *GoBug.ini*. Notez que si vous avez renommé GoBug, le fichier *ini* aura aussi le nouveau nom.

"Sorry there was an error starting GoBug during starting procedure No. ....."
"Sorry, the system did not respond correctly to GoBug during procedure No ....."

"Désolé, il y a eu une erreur de démarrage de GoBug pendant la procédure de démarrage n °....."
"Désolé, le système n'a pas répondu correctement à GoBug pendant la procédure n °....."

Cause : Ceci est un message du gestionnaire d'exceptions de GoBug montrant que quelque chose s'est mal passé avec GoBug. J'espère que vous ne verrez jamais ces messages !

3.6.2 Problèmes lors du démarrage du debuggee

"Sorry, file not found" *or*"Sorry, path not found"

"Désolé, fichier non trouvé" ou "Désolé, chemin introuvable"

Cause : Un fichier ou un chemin d'accès a été spécifié qui n'a pas pu être trouvé lorsque GoBug a essayé de le déboguer.

"Sorry, access to this file denied"

"Désolé, il y a eu une erreur de démarrage de GoBug pendant la procédure de démarrage n °....."

Cause : Un fichier ou un chemin d'accès a été spécifié qui n'a pas pu être trouvé lorsque GoBug a essayé de le déboguer.

"Sorry, cannot debug this file. Is it a 32-bit executable?"

"Désolé, impossible de déboguer ce fichier. Est-ce un exécutable 32 bits ?"

Cause : GoBug a constaté que le debuggee proposé n'était pas un exécutable au format Windows 32 bits PE. GoBug ne peut pas déboguer un exécutable qui n'est pas dans ce format. Ce message d'erreur apparaîtra également si vous essayez de charger une DLL à partir de la ligne de commande. Une DLL ne peut être déboguée qu'en chargeant d'abord en tant que debuggee le fichier exe qui utilise cette DLL.

"The PE signature was not found in the debuggee image."

"La signature PE n'a pas été trouvée dans l'image du debuggee."

Cause : L'exécutable qui a été chargé (en tant que debuggee) a semblé être un exécutable Win32, mais sa signature PE était manquante suggérant qu'il était corrompu ou qu'il n'était pas, en réalité, un fichier Win32.

"The PE signature was not found in a dll or loaded process."

"La signature PE n'a pas été trouvée dans un DLL ou le processus chargé."

Cause : Une Dll chargée par le debuggee ou un EXE créé par le debuggee a été identifié comme n'ayant pas de signature PE suggérant qu'il (elle) est corrompu(e) ou n'est pas, en fait, un fichier Win32.

"There is problem getting information about this debuggee."

"Il y a un problème pour obtenir des informations sur ce debuggee."

Cause : L'exécutable qui a été chargé semble être un exécutable Win32, mais ne semble pas tout à fait dans le bon format. Il est probablement corrompu d'une manière ou d'une autre.

"There is problem getting information about a dll or loaded process."

"Il y a un problème pour obtenir des informations sur ce debuggee."

Cause : Une DLL chargée par le debuggee ou un EXE créé par le debuggee n'a pas été trouvée dans le bon format Win32. Il s'agit probablement de fichiers corrompus d'une manière ou d'une autre.

"Could not find symbols for the debuggee. GoBug looks for COFF or CodeView symbols embedded in the image. It can also get CodeView symbols from a pdb file, COFF symbols from a DBG file, or symbols from a MAP file (Borland or Microsoft type). Check your linker settings, or for the time being you can proceed without symbols".

"Impossible de trouver des symboles pour le debuggee. GoBug recherche les symboles COFF ou CodeView incorporés dans l'image. Il peut également obtenir les symboles CodeView d'un fichier pdb, les symboles COFF d'un fichier DBG ou les symboles d'un fichier MAP (type Borland ou Microsoft). Vérifiez les paramètres de votre éditeur de liens ou, pour le moment, vous pouvez continuer sans symboles "

Cause : Explicite, mais voir néanmoins le paragraphe GoBug comme un débogueur symbolique.

"Debug information was wrong format. Check linker is giving the correct output. For the time being you can proceed without symbols."

"Les informations de débogage sont dans un format incorrect. Vérifiez que l'éditeur de liens donne la bonne sortie et, pour l'instant, vous pouvez continuer sans symboles."

Cause : Dans ce cas, certaines informations de débogage ont été trouvées (incorporées ou dans un fichier dbg, pdb ou map) mais elles étaient illisibles car libellées dans le mauvais format ou corrompues. Voir le paragraphe GoBug comme un débogueur symbolique.

"GoBug found a map file made at 08:30 on 23rd February 2004 but Testbug.exe was made at 09:30 on 23rd February 2004. Do you want to use this map file bearing in mind it was not made at the same time?"

"GoBug a trouvé un fichier map fait le 23 février 2004 à 08:30 mais Testbug.exe a été fait le 23 février 2004 à 09:30. Voulez-vous utiliser ce fichier map en gardant à l'esprit qu'il n'a pas été fait en même temps ? "

Cause : Au chargement, un fichier map a été trouvé qui ne semblait pas correspondre à l'exécutable en cours de chargement, car il avait le mauvais horodatage. (Notez que les données et l'heure indiquées dans le message seront ajustées pour tenir compte du paramètre d'heure d'été présent sur votre ordinateur) Voir Procédures de recherche pour les fichiers map.

3.6.3 Problèmes avec l'interface utilisateur de GoBug

>> When moving Gobug's main window or inspectors the background is not repainted

Lorsque l'on déplace la fenêtre principale ou les inspecteurs de Gobug, l'arrière-plan n'est pas repeint

Cause : Si GoBug est au-dessus du debuggee et que l'exécution du debuggee est suspendue dans la boucle de débogage (ou parce que vous consultez le journal), le debuggee *ne sera pas* repeint. Si ce n'est pas le cas, la fenêtre en-dessous de GoBug peut ne pas répondre pour une raison ou une autre. En mode mono-utilisateur Windows, vous pouvez voir si c'est le cas en appuyant *une fois seulement* sur Ctrl+Alt+Suppr (sinon utilisez le Gestionnaire des tâches). Une fenêtre apparaît montrant toutes les fenêtres dont le système a connaissance et qui ne répondent pas (c'est-à-dire qui ne reviennent pas de la procédure de fenêtre). Une autre cause possible est la suivante. Windows envoie un message WM_PAINT à toutes les fenêtres découvertes dans une opération de déplacement. La fenêtre non couverte doit répondre au message dans sa procédure de fenêtre en se redessinant. Si la fenêtre découverte essaie de dessiner sur une partie quelconque de l'écran qui n'est pas "invalide", un tel dessin sera ignoré par le système. Il se peut que le système ait perdu la trace de la zone de l'écran qui est invalide.

>> The debuggee window tries to come to the foreground but is improperly painted

La fenêtre de débogage essaie de venir au premier plan mais est mal peinte

Cause : Cela ainsi que d'autres effets visuels similaires peuvent être causés par la suspension du debuggee dans la boucle de débogage lors de l'exécution des messages de fenêtre. Par exemple, l'action que le debuggee prendrait s'il était libre de le faire pourrait être d'apparaître au premier plan. Cependant, il ne peut pas le faire parce qu'il est suspendu. Si vous effectuez des breaks de message en pas-à-pas sur divers messages, l'action du debuggee sera complétée pas par pas.

>> Zone Alarm Pro triggers when GoBug's hot key is pressed, or at other times.

Zone Alarm Pro (anti-virus) se déclenche lorsque vous appuyez sur la touche de raccourci de GoBug, ou à d'autres moments.

Cause : Zone Alarm Pro a probablement détecté un effort de GoBug pour se placer au premier plan.

Solution : Désactivez l'alerte en accédant à Zone Alarm, Program Control/Programs, puis définissez la colonne Access/Trusted pour GoBug.exe à Allow (Autoriser).

>> My Unicode section names don't appear correctly.

Mes noms de section Unicode n'apparaissent pas correctement.

En utilisant l'assembleur GoAsm, vous pouvez libeller un nom de section en utilisant des caractères non-romains si vous utilisez un éditeur Unicode. Les outils "Go" stockent les noms des sections dans le fichier objet et dans l'exécutable au format UTF-8. Dans l'exécutable, cependant, il y a seulement de la place pour les noms de section de 8 caractères. Donc, si vous avez utilisé des caractères non-romains, il y a une forte possibilité pour que les noms de section (au format UTF-8) excèdent l'espace disponible de 8 caractères. Le résultat est que ces noms seront tronqués et que le dernier caractère visible sera corrompu.

>> My Unicode characters do not appear properly in my dumps.

Mes caractères Unicode n'apparaissent pas correctement dans mes vidages

En utilisant les outils "Go", vous pouvez utiliser des caractères non-romains pour les labels de code et de données (symboles). Ceux-ci ne s'afficheront correctement dans vos vidages que si vous spécifiez le vidage dans un fichier UTF-8 ou UTF-16 et affichez le fichier résultant à l'aide d'un éditeur Unicode.

3.6.4 Problèmes pendant le débogage

>> My application uses Structured Exception Handling for signalling and the exception dialog keeps appearing.

Mon application utilise la gestion d'exception structurée (SEH ou *Structured Exception Handling) pour la signalisation et la boîte de dialogue d'exception continue d'apparaître.*

Cause : La gestion des exceptions structurées (SEH *ou* Structured Exception Handling) peut être utilisée par une application comme signal, par exemple, si plus de mémoire est requise. Dans ce cas, une exception de lecture/écriture peut se produire et le gestionnaire d'exceptions est censé fournir la mémoire supplémentaire requise.

Solution : Il n'y a pas de solution à cela puisque GoBug ne peut pas faire la différence entre les exceptions intentionnelles et les erreurs d'une application. Lorsqu'une telle exception se produit, assurez-vous de cliquer sur le second bouton radio dans la boîte de dialogue d'exception, ce qui permet au debuggee de traiter l'exception elle-même. Notez que le système Windows utilise souvent la SEH pour la signalisation et les tests. J'ai trouvé que de telles exceptions se produisent pendant les appels à HHCTRL.OCX (pour visualiser l'aide), et également dans d'autres exemples, par exemple dans l'API *IsBadReadPtr*. Les exceptions système provoquées de cette manière sont en fait envoyées à GoBug mais GoBug les ignore et ne les enregistre pas.

>> I know the debuggee is loading a Dll but I cannot see it in the log, nor in the "Information about debuggee" pane.

Je sais que le debuggee est en train de charger une DLL, mais je ne la vois pas dans le journal ni dans le volet "Information about debuggee" ("Informations sur le debuggee ").

Cause : Cela peut arriver si le nom de la DLL a été changé. Quand une DLL est chargée, GoBug en est informé par le système, mais GoBug n'a pas le nom ou le chemin d'accès de la DLL. GoBug doit trouver le nom en regardant à l'intérieur de la DLL dans le répertoire d'exportation (qui est le seul endroit à l'intérieur de la DLL où le nom est disponible). Le nom dans le répertoire d'exportation est inséré par l'éditeur de liens lors de la compilation et sera donc le nom d'origine de la DLL, et non le nom modifié. Parce que le nom que GoBug obtient pour la DLL sera faux, il sera impossible de trouver le chemin d'accès pour le fichier non plus. La même chose peut arriver dans le cas improbable d'une DLL qui n'a pas d'exportations (et donc pas de répertoire d'exportation !).

"Sorry, access denied by the system."

"Désolé, accès refusé par le système."

Cause : Lors de la création d'un inspecteur, il a été constaté que le système n'autoriserait pas l'inspection de la zone de mémoire choisie. Souvenez-vous que les adresses mémoire sont toutes référencées dans le contexte mémoire du debuggee ou dans la mémoire partagée. Il est possible que l'adresse mémoire que vous essayez d'inspecter ne soit pas attribuée par le système ou qu'elle ne soit pas lisible.

"Sorry, access no longer available."

"Désolé, l'accès n'est plus disponible."

Cause : Ce message se trouve dans les inspecteurs de données lorsque le système n'autorise plus l'accès à la zone de mémoire couverte par l'inspecteur de données. Cela se produit lorsque la zone de mémoire a été libérée par le debuggee, par exemple en appelant *VirtualFree*.

"Sorry no symbols were loaded."

"Désolé, aucun symbole n'a été chargé"

"Sorry since no symbols were loaded, you cannot specify that you want to view one!"

"Désolé car aucun symbole n'a été chargé. Vous ne pouvez pas spécifier que vous voulez en voir un !"

"Sorry since no symbols were loaded you cannot specify a symbol to run to."

"Désolé, car aucun symbole n'a été chargé. Vous ne pouvez pas spécifier de symbole à utiliser."

Voir le paragraphe GoBug comme un débogueur symbolique.

"Sorry, GoBug does not know this data symbol."

"Désolé, GoBug ne connaît pas ce symbole de donnée."

"Sorry, GoBug does not know this code symbol or procedure."

"Désolé, GoBug ne connaît pas ce symbole de code ou cette procédure."

Cause : L'un de ces messages peut apparaître si, dans la boîte de dialogue de procédure Exécuter pour... ou Afficher..., vous avez entré un symbole qui n'est pas reconnu par GoBug. Cela peut arriver si vous avez fait une erreur. Cela peut également arriver si un inspecteur de symboles ou un point d'arrêt a été utilisé dans une session précédente avec le même débogueur, et que vous essayez d'utiliser le même inspecteur ou le même point d'arrêt alors que le symbole est manquant dans cette session. Le symbole peut être manquant si le debuggee a maintenant été passé à l'éditeur de liens sans demande d'informations de débogage. Il peut également être manquant si vous avez supprimé le symbole concerné du code source. Voir la section sessions

>> Some of my symbols containing the '@' character do not appear properly.

Certains de mes symboles contenant le caractère '@' n'apparaissent pas correctement.

Cause : Certains outils de développement "enrichissent" certains symboles en utilisant un trait de soulignement et '@' suivi d'un nombre décimal. Ceci indique aux outils que le symbole accepte des paramètres (le nombre décimal donne le nombre d'octets dans les paramètres). Lors de l'affichage de ces symboles, GoBug supprime cet enrichissement, mais cela signifie que si vous utilisez '@' dans vos noms de symboles suivis d'un nombre décimal, ces caractères n'apparaîtront pas dans la table des symboles de GoBug.

"No exact match, do you mean .....?"

"Pas de correspondance exacte, voulez-vous dire ...?"

Cause : Vous êtes en train de saisir un nom de symbole dans une boîte de dialogue mais le symbole n'est pas reconnu par GoBug. On vous demandera si vous vouliez entrer le symbole que GoBug pense que vous auriez pu signifier à la place.

"Sorry cannot disassemble at present".

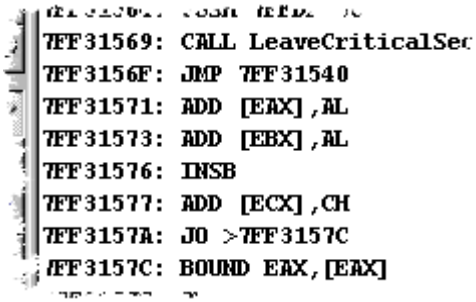
"Désolé. Ne peut pas désassembler à présent"

Cause : Ce message apparaît dans le volet de code si l'instruction en cours du thread affiché semble être en dehors de toute section de code connue, ou si la zone de mémoire concernée est illisible.

>> Some lines of the disassembly show the wrong mnemonics.

Certaines lignes du désassemblage font apparaître des mnémoniques aberrants.

Cause : Cela peut arriver si le compilateur utilisé pour faire l'EXE a inséré des zéros pour le remplissage. Si cela se produit, GoBug ne saura pas si le zéro fait partie d'un opcode ou non. Voici un exemple de ce qui peut arriver :



Solution : Demandez au compilateur de combler avec des NOP (opcode 90) à la place.

"You must enter a 64-bit hex number eg. 54100A23842F922Ch."

"Vous devez entrer un nombre hexadécimal de 64 bits, par exemple 54100A23842F922Ch."

"You must enter a 32-bit hex number, eg. 54100A23h or 842F922Ch."

"Vous devez entrer un nombre hexadécimal de 32 bits, par exemple 54100A23h ou 842F922Ch."

"You must enter a 16-bit number eg. 13F0 or 21B4"

"Vous devez entrer un nombre de 16 bits, par exemple 13F0 ou 21B4"

Cause : Ces messages sont typiques de messages d'erreur ayant trait à une saisie de nombres. Lorsque vous entrez des nombres hexadécimaux, supprimez d'abord les chiffres existants, puis insérez les nouveaux chiffres. Le "h" est optionnel. Dans certains cas, les espaces sont ignorés, par exemple lors de la modification de registres MMX ou XMM.

"You must enter a decimal or real number eg. 2,001 or 3.142 or 22.625E4."

"Vous devez entrer un nombre décimal ou réel, par exemple 2,001 ou 3.142 ou22.625E4."

"Exponent must be between -4932D and +4932D"

"L'exposant doit être entre –4932D et +4932D"

"That is an invalid exponent - it must be a decimal integer"

"L'exposant est invalide – il doit être sous la forme d'un entier décimal"

"The number is too small to be resolved into the floating point register"

"Le nombre est trop petit pour être résolu dans le registre à virgule flottante"

"The number is too big for the floating point register".

"Le nombre est trop grand pour le registre en virgule flottante"

Cause : Ces messages peuvent être rencontrés si des erreurs affectent la modification du contenu des registres à virgule flottante. Le "D" à la fin du message de l'exposant souligne que le nombre doit être en décimal, mais vous n'avez pas besoin d'entrer ceci. Voir les paragraphes nombres réels ou Modifications des registres à virgule flottante.

"This is not a recognised handle or 16-bit selector".

"Ce n'est pas un handle reconnu ou un sélecteur 16 bits"

Cause : Vous avez été invité à saisir un handle ou un sélecteur 16 bits mais le système a signalé qu'il n'était pas reconnu. Ce test est effectué à l'aide de l'API *GetThreadSelectorEntry*. Rappelez-vous que cela ne fonctionne que pour les adresses de mémoire dans le contexte de la mémoire propre du debugee ou dans la mémoire partagée.

"You are about to step into a system API or other code in shared memory run by the system. This is normally not permitted and may cause problems, for example, a. there may be a thunk to 16-bit code which GoBug cannot follow. b. the system may switch contexts so you may lose the trap flag. c. The system may close the debuggee if it senses your presence. Do you wish to proceed?"

"Vous êtes sur le point d'entrer dans une API système ou un autre code dans la mémoire partagée exécutée par le système, ce qui est normalement interdit et peut causer des problèmes. Par exemple, a) cela peut être un avatar 16 bits que GoBug ne peut pas résoudre. b) le système peut changer de contexte pour que vous perdiez le flag de trap. c) Le système peut fermer le debuggee s'il détecte votre présence. Voulez-vous continuer ?

Cause : Vous utilisez GoBug avec Windows 9.xx, vous avez appuyé sur F5 (mode pas-à-pas) et l'instruction suivante entraînera l'exécution dans une DLL système. Lorsque vous utilisez GoBug avec Windows NT/2000/XP, ce message n'apparaît pas. Évidemment, avant de passer à une DLL système ou à un produit tiers protégé par un droit d'auteur, vous devez avoir la permission légale de le faire. Consultez la section Mode pas-à-pas dans le code Windows.

"GoBug is trying to intervene in the execution of the debuggee but some of its code appears to be shared by another process! Bearing in mind the other process might be affected do you wish to proceed?"

"GoBug essaie d'intervenir dans l'exécution du débuggee mais une partie de son code semble être partagée par un autre processus ! En gardant à l'esprit que l'autre processus pourrait être affecté, voulez-vous continuer ?"

Cause : Vous avez probablement cherché à obtenir un thread en cours d'exécution du debuggee dans la boucle de débogage en utilisant le contrôle de feux tricolores, et GoBug inonde le code de points d'arrêt pour essayer d'y parvenir. Au cours de ce processus, GoBug a découvert qu'une section de code avait des caractéristiques «partagées» et que GoBug ne voulait pas provoquer le break d'un autre exécutable sur le point d'arrêt qu'il insérait sans vous le demander préalablement. Voir la section Reprise du contrôle sur les threads récalcitrants.

"Breakpoint not allowed in code areas shared by another process."

"Le point d'arrêt n'est pas autorisé dans les zones de code partagées par un autre processus."

Cause : Vous essayez de définir un point d'arrêt dans une zone de code que GoBug a identifiée comme étant partagée par un autre processus. Sous Windows 9.xx, le test effectué par GoBug est simplement de savoir si la zone de code est en mémoire partagée (au dessus de 7FFFFFFFh). Sous NT/2000/XP, des tests plus sophistiqués sont nécessaires.

"Please first press F5/6 to move the debuggee past the proposed breakpoint."

"Veuillez d'abord appuyer sur F5/6 pour positionner le debuggee au-delà du point d'arrêt proposé."

Cause : Vous essayez de définir un point d'arrêt à l'EIP courant. GoBug est incapable de sauter par-dessus l'EIP courant pour définir le point d'arrêt, car il ne peut pas être certain qu'il gardera le contrôle du debuggee. Vous devrez effectuer une seule étape pour amener le débogueur sur l'instruction suivante. Ensuite, vous pouvez définir le point d'arrêt.

"GoBug tried, but could not trap the following threads .....".

"GoBug a essayé, mais n'a pas pu piéger les threads suivants ...".

Cause : Vous avez probablement cherché à obtenir un thread en cours d'exécution du debuggee dans la boucle de débogage en utilisant le contrôle à feux tricolores, et GoBug a inondé le code de points d'arrêt pour tenter d'y parvenir. Cependant, cette action a échoué dans le cas des threads répertoriés. Cela pourrait probablement être dû au fait que ces threads sont suspendus ou actuellement dans une API système qui n'a pas encore été retournée. Voir Reprise du contrôle sur les threads récalcitrants.

"Tried, but failed, to trap a thread".

"Essayé, mais a échoué, pour piéger un thread"

Cause : Ce message apparaît si vous appuyez sur la touche de raccourci et que GoBug n'a pas réussi à placer les threads du debuggee dans la boucle de débogage. Voir Reprise du contrôle sur les threads récalcitrants.

"No action taken by GoBug (debuggee already waiting in debug loop)"

"Aucune action engagée par GoBug (le debuggee déjà en attente dans la boucle de débogage)"

Cause : Vous avez probablement cherché à obtenir un thread en cours d'exécution du debuggee dans la boucle de débogage en utilisant le contrôle à feux tricolores. Cependant, GoBug ne l'a pas fait parce qu'un thread est déjà dans la boucle de débogage. En conséquence, tout le processus du debuggee et tous ses threads ont été suspendus. Voir Reprise du contrôle sur les threads récalcitrants.

"The trap for this thread has been released. Try setting a breakpoint to trap it (or try the hot-key)."

"Le flag de trap de ce thread a été libéré. Essayez de définir un point d'arrêt pour le piéger (ou essayez le raccourci clavier)."

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais le thread actuellement affiché dans le volet de code est en cours d'exécution sans piège (action F8 ou F9). Par conséquent, GoBug n'a pas pu répondre à l'action sur la touche. Voir successivement Reprise du contrôle sur les threads récalcitrants, Insertion de votre propre point d'arrêt de code, la touche de raccourci.

"The thread currently showing no longer exists!"

"Le thread en cours d'affichage n'existe plus !"

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais le thread actuellement affiché dans le volet de code est terminé. À défaut de pas-à-pas ou de l'atteinte d'un point d'arrêt, il est normal que l'affichage continue d'afficher un thread terminé (bien qu'il soit grisé). Voir le débogage des applications multithread.

"The thread currently showing is in a procedure which has not yet returned:- ....."

"Le thread en cours d'affichage est dans une procédure qui n'a pas encore retourné : ....."

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais GoBug n'a pas pu lui répondre car le thread n'est pas dans la boucle de débogage. Ce message particulier est donné si le thread concerné n'est pas encore retourné d'une procédure connue de GoBug. L'EIP de l'instruction CALL est donné (si connu) et le volet de code montrera l'appel. Normalement, vous verrez également la flèche EIP animée. Voir le paragraphe Boucle de débogage.

"The thread currently showing has not yet returned from the window procedure"

"Le thread en cours d'affichage n'est pas encore revenu de la procédure de fenêtre"

Cause : Vous avez probablement appuyé sur l'une des touches «action» (F5 à F9) tout en ignorant un message ayant appuyé sur F6 après un break de message pas-à-pas. Tout en ignorant un message, GoBug conserve le contrôle du thread en définissant le trap dans la boucle de débogage lors de chaque instruction. Cela ralentit le débogage et s'il a beaucoup de travail à faire sur un message particulier, il peut y avoir un certain délai avant que le message ne soit traité.

"The thread currently showing is in code at EIP ....."

"Le thread en cours d'affichage est dans le code à EIP = ..."

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais GoBug n'a pas pu répondre à cette touche car le thread n'est pas dans la boucle de débogage. Ce message particulier est donné si le thread concerné n'a pas encore retourné à partir de certains codes, mais GoBug ne sait pas dans quelle procédure il se trouve. Voir le paragraphe Boucle de débogage.

"The thread currently showing is currently in GoBug's hook procedure."

"Le thread affiché est actuellement dans la procédure de hook de GoBug."

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais GoBug n'a pas pu répondre à cette touche car le thread n'est pas dans la boucle de débogage. Ce message particulier est donné si le thread concerné est dans le code dans GoBug.dll (qui contient la procédure de hook de GoBug dans l'espace d'adressage du debuggee). Cela signifie généralement que GoBug travaillait toujours sur l'événement ou le message lorsque vous avez appuyé sur la touche. Voir les paragraphes Boucle de débogage, Comment GoBug surveille les événements.

"GoBug has no idea what happened to the thread currently showing."

"GoBug n'a aucune idée de ce qui est arrivé au thread affiché actuellement."

Cause : Vous avez probablement appuyé sur l'une des touches "action" (F5 à F9) mais GoBug n'a pas pu répondre à cette touche car le thread n'est pas dans la boucle de débogage. Ce message particulier est donné si les informations sur le thread ne peuvent pas être invoquées.

"Please inform JG@JGnet.co.uk as follows:

GoBug Version ..... At Eip=..... Could not read memory at ...../Unexpected result 103/16-bit Kernel could not be found".

"Veuillez informer JG@JGnet.co.uk comme suit :
GoBug Version ..... à Eip = ... Impossible de lire la mémoire à ..... / Résultat inattendu 103/Le Kernel 16 bits n'a pas pu être trouvé".

Cause : Ce sont les messages du gestionnaire d'exceptions de GoBug appelés lors d'une tentative d'obtenir des informations sur le debuggee, éventuellement pendant une promenade. Ce ne sont pas des messages fatals et, bien qu'il y ait eu un problème lors de la collecte d'informations, cela a été anticipé et aura été avalé par le gestionnaire.

"There are too many "finds". The search needs to be more specific."

"Il y a trop d'éléments trouvés. La recherche doit être mieux ciblée."

Cause : Lors de la recherche dans mémoire partagée ou celle du debuggee, la boîte de résultats était en danger de surcharge en raison du nombre de "hits". Le nombre maximum autorisé est de 5 000. Voir le paragraphe Recherche en mémoire…

>> I gave my sections names, but they appear truncated.

J'ai donné des noms à mes sections, mais ces noms apparaissent tronqués.

Cause : Il est important de rappeler ici que les spécifications du fichier PE n'autorisent pas les noms de section de plus de 8 octets. Si les noms utilisent des caractères latins (par exemple, l'anglais), la longueur maximale est de 8 caractères. Si les noms utilisent des caractères non romains, ils peuvent être tronqués encore plus, car ils doivent être stockés dans l'exécutable en Unicode, ce qui prend plus de place.

>> I cannot amend the MMX, 3DNow! or XMM registers

Je ne peux pas modifier les registres MMX, 3DNow! ou XMM

Cause : Cette anomalie est probablement due au fait que les registres que vous essayez de modifier ne sont pas disponibles sur votre processeur. Une ligne d'avertissement devrait apparaître dans le volet de registre pour vous le dire. Voir le paragraphe Volets de registre spéciaux (MMX, FPU, 3DNow !, XMM, SSE et SSE2) pour voir quels registres sont disponibles sur quels processeurs.

>> You are changing the contents of a floating point register in the FPU by hand but the "value" still shows as empty.

Vous modifiez manuellement le contenu d'un registre à virgule flottante dans la FPU, mais la "valeur" est toujours vide.

Cause : Le mot-clé est défini sur "vide" pour le registre concerné.

Solution : Remplissez le registre soit en chargeant un nombre normalement (en effectuant un pas simple par-dessus une instruction de chargement FP), soit en changeant le contenu du registre MMX correspondant pour changer ledit mot-clé.

"Exponent cannot be larger than +4001h or smaller than -3FFEh."

"L'exposant ne peut pas être supérieur à +4001h ou inférieur à –3FFEh."

Cause : Vous êtes en train de modifier le contenu des registres à virgule flottante à la main, mais vous avez libellé un exposant trop grand ou trop petit. Voir le paragraphe Modification des registres à virgule flottante en ce qui concerne l'usage de valeurs pertinentes dans la FPU. Des valeurs différentes seront appliquées dans les registres à virgule flottante 3DNow !, SSE et SSE2.

"Full register data has been lost."

"Les données du registre complet ont été perdues."

Cause : Lorsque vous essayez d'afficher l'état des registres dans le journal, GoBug n'a pas pu trouver les informations requises. Pour économiser de la mémoire, GoBug conserve uniquement les modifications apportées aux registres du journal. Lorsque les registres sont affichés, GoBug doit retourner aux dernières valeurs de registre connues pour compléter toutes les informations nécessaires pour afficher l'état des registres à un moment donné. Ce message peut très bien apparaître si l'enregistrement précédent a été supprimé, par exemple si vous avez effacé le journal à l'aide de l'élément de menu "settings, clear log" ("Paramètres, effacer le journal"). Cela peut se produire si l'enregistrement antérieur a fait une boucle et que le thread s'est terminé.

Solution : Visualisez le journal avant le thread auquel vous vous intéressez. Vous pouvez également augmenter la quantité de mémoire utilisée par le journal pour empêcher les enregistrements antérieurs de disparaître trop rapidement. Vous pouvez également activer un point d'arrêt au début du code qui vous intéresse. GoBug crée alors un enregistrement complet dans le journal de tous les registres. Voir les paragraphes Le journal de bord les volets correspondants ou Débogage utilisant le journal.

"part of the record looped away"

"Une partie de l'enregistrement a disparu"

Vous pouvez trouver cette mention dans le journal

Cause : Cela peut apparaître dans le journal si la mémoire du journal est pleine. Dans ce cas, les enregistrements antérieurs ont dû être abandonnés pour faire de la place aux enregistrements ultérieurs. Ce n'est normalement pas un problème, mais vous pouvez réduire la quantité de mémoire de journal utilisée en cours d'exécution à la zone de code à laquelle vous vous intéressez à l'aide de points d'arrêt. Cela est dû au fait que rien n'est consigné lors de l'exécution d'un point d'arrêt. Voir les paragraphes Le journal de bord les volets correspondants ou Débogage utilisant le journal.

"Sorry, could not establish memory areas - memory resources must be low."
"Floating point pane needs 4K of memory but the system could not provide this."

"Désolé, impossible d'établir les zones de mémoire – les ressources mémoire doivent être faibles."
"Le volet virgule flottante nécessite 4 Ko de mémoire, mais le système n'a pas pu les fournir."

Cause : Ces messages sont typiques d'une situation de mémoire insuffisante ou d'un contexte où le système ne sert plus GoBug. Dans une telle situation, fermez les applications, sauvegardez le travail et redémarrez.

"Sorry, could not create the file."

"Désolé, impossible de créer le fichier."

Cause : Ce message est typique des messages d'erreur de création de fichier lorsque vous tentez d'effectuer un vidage vers un fichier ou d'extraire des ressources. Les causes possibles sont : disque plein, nom de fichier identique à un autre fichier ou chemin d'accès inexistant.

"GoBug could not perform this task."

"GoBug n'a pas pu effectuer cette tâche."

Cause : Ceci est un message de caractère général qui s'affiche dans le cas où GoBug n'a pas pu faire quelque chose qu'on lui a demandé. Comme il est réservé aux événements inhabituels pour lesquels une explication n'est pas spécialement requise, aucun autre détail n'est donné dans le message.

>> The single-step message break is not cutting in on messages which I know are being sent.

Le break de message en pas-à-pas ne coupe pas les messages que je sais être envoyés.

Essayez :

Assurez-vous que le break de message en pas-à-pas est activé pour le thread que vous traitez. Vous pouvez vérifier cela en utilisant le contrôle de feux tricolores. Vous pouvez également activer et désactiver le saut de message en une seule étape pour le thread principal ou pour les nouveaux threads à partir du menu "actions". Notez également que le saut de message en une seule étape ignore les messages ignorés par le journal. C'est par conception et les ignorants de journal peuvent être modifiés en utilisant l'élément de menu "paramètres, paramètres de journal" et l'onglet "ignore".

"The message you are running to occurred but it was going to shared memory and could not be trapped"

"Le message que vous exécutez a eu lieu mais il allait à la mémoire partagée et ne pouvait pas être piégé"

Cause : Certains messages sont envoyés uniquement aux DLLs système dans la mémoire partagée sauf si vous avez sous-classé la classe de fenêtre. Les exemples sont TB_ADDBUTTON (qui ajoute un bouton à une barre d'outils) ou LB_ADDITEM (qui ajoute un élément à une liste). Votre application enverra ces messages à la fenêtre de contrôle commune concernée, et la procédure de fenêtre sera dans une DLL système. Étant donné que les DLL système sont en mémoire partagée (dans W95 et W98 en tout cas), vous ne pouvez pas piéger le message. C'est parce que si vous essayiez de le faire, cela pourrait également interrompre d'autres applications.

>> When printing some lines are missing from the printed page or the paper is too narrow for the printed material.

Lors de l'impression, certaines lignes sont absentes de la page imprimée ou le papier est trop étroit pour le document imprimé.

Cause : Lorsque vous êtes sur le point d'imprimer, GoBug récupère les informations relatives à l'imprimante et au format de papier du système. Si du papier est manquant sur les bords du papier, le papier que vous utilisez n'est probablement pas de la même taille que celle spécifiée dans la boîte de dialogue "Propriétés" de l'imprimante.

Solution : En utilisant le menu Démarrer, allez dans "settings, printers" ("paramètres, imprimantes"), faites un clic droit sur l'imprimante par défaut et vérifiez dans "propriétés" que le format de papier spécifié est le même que le papier que vous utilisez réellement.

3.6.5 Problèmes au moment de la fermeture du debuggee

"The debuggee has closed. Click OK to close the panes."

"Le debuggee s'est fermé.Cliquez sur OK pour fermer les volets."

Cause : Le debuggee a été fermé (soit sa fenêtre principale est fermée, soit il a terminé son traitement et a appelé ExitProcess, ou a été renvoyé au système). Si cela était inattendu, il est possible que le système ait rejeté le debuggee. Cela peut se produire, par exemple, si vous essayez de faire défiler le code système. Voir le paragraphe Pas-à-pas dans le code Windows.

"Could not close the debuggee in the usual way there may be system conflicts. Ensure you save all work in other applic ations before closing this message box."

"Impossible de fermer le debuggee de la manière habituelle. Il peut y avoir des conflits système.Veillez à sauvegarder tout le travail dans d'autres applications avant de fermer cette boîte de message."

Cause : GoBug fait tous les efforts pour fermer proprement le debuggee mais ce n'est pas toujours possible. Si cette boîte de message apparaît, il est possible que les ressources du système deviennent excessivement surchargées ou qu'une erreur cachée puisse survenir. Si vous effectuez des actions critiques, il peut être judicieux de redémarrer après avoir vu ce message. Voir le paragraphe Comment GoBug ferme le debuggee.

3.6.6 Problèmes fatals

"Sorry there was a serious error and GoBug version .... will have to close. It will try to do this without affecting other parts of the system. There was exception ..... at eip=..... Please inform Jeremy Gordon on JG@JGnet.co.uk and get the latest version of GoBug from http://www.GoDevTool.com (or from http://www.GoProg.com)"

"Désolé, une erreur grave est survenue et GoBug version..... devra fermer. Il va essayer de le faire sans affecter les autres parties du système. Il y a eu une exception….. à EIP = ... Veuillez en informer Jeremy Gordon sur JG@JGnet.co.uk et télécharger au besoin la dernière version de GoBug à partir de http://www.GoDevTool.com (ou de http: / /www.GoProg.com) "

Cause : Ceci est un message du gestionnaire d'exceptions de GoBug. Une erreur est survenue dans GoBug. C'est probablement un bug qui a échappé aux tests au cours du développement. Pardon ! En tant que développeur, vous comprendrez. Tous les efforts raisonnables ont été faits pour éradiquer les bogues.

3.7 Messages d'erreur et dépannage

GoBug Debugger Help
GoBug Version 2.03.01 (for 9x, ME, NT, 2000, XP, Vista and Windows7)

Chapitre 4 – Écriture de programmes en Unicode

Ce chapitre est destiné aux personnes intéressées par l'écriture de programmes Unicode (et, plus généralement, des programmes utilisant des caractères non latins) pour Windows 32 bits, en utilisant GoAsm (assembleur), GoRC (compilateur de ressources) et GoLink (éditeur de liens). Il pourra également intéresser ceux qui utilisent d'autres outils ou d'autres langages de programmation.

4.1 Introduction

4.1.1 Qu'est-ce que Unicode ?

Unicode est une norme de codage de caractères qui vise à couvrir toutes les grandes écritures du monde. Elle est publiée par le Consortium Unicode, une organisation sans but lucratif basée à Mountain View, en Californie, dont les membres comprennent des sociétés informatiques, des éditeurs de logiciels, des universitaires et autres personnes intéressées par le sujet.
Les développeurs n'auront probablement pas besoin de se plonger dans la norme parce que tout le codage est obtenu grâce à des outils tels que les éditeurs Unicode. Mais pour ceux qui sont intéressés, il y a beaucoup d'informations à ce sujet sur le site Web Consortium Unicode. La norme a été publiée en 1991 dans *The Unicode Standard*. Elle est régulièrement mise à jour.
Actuellement, environ 50 000 caractères de plus de 90 scripts sont couverts par la norme, et il y a beaucoup de place pour prendre en charge des expansions ultérieures.

La caractéristique la plus marquante de l'Unicode est que les caractères sont codés sur 16 bits au lieu des 8 bits de la norme ASCII. Il en résulte qu'un caractère peut, soit prendre une valeur comprise entre 0 et 0FFFFh (le cas habituel), soit en utilisant des paires de substitution, être représentés par deux de ces valeurs. Pour cette raison, les caractères Unicode sont parfois décrits comme des caractères "larges". Cette caractéristique demeure présente même dans l'UTF-8 et autres formats Unicode qui ne sont que la traduction des valeurs de caractères.

Ce tableau sommaire donne un aperçu de l'étendue de la couverture Unicode :

Scripts Généraux	0000 à 1FFF	Latin (Romain) et autres scripts non-idéographiques, par exemple, le Grec, le Cyrillique, l'Arménien / Hébreu, l'Arabe, l'Oriya / Tamil, le Thai / Lao, le Tibétain, le syllabaire autochtone canadien, le Mongol
Symboles	2000 à 2DFF	Par exemple, les monnaies, nombres, maths, les flèches, les symboles techniques, les modèles braille, dingbats et codes de contrôle bidirectionnels
Caractères phonétiques et symboles CJC[6]	2E00 à 33FF	Caractères phonétiques, signes de ponctuation et symboles utilisés en Chinois, Japonais et Coréen
Idéogrammes CJC	3400 à 9FFF	Caractères idéographiques Han unifiés à partir de sources chinoises, japonaises et coréennes
Syllabes Yi	A000 à A4CF	Syllabes Yi et radicaux Yi
Syllabes Hangul	AC00 à D743	Syllabes précomposées coréennes Hangul
Substituts	D800 à DFFF	Indique que le caractère est représenté par deux caractères 16 bits
Zone à usage privé	E000 à F8FF	Réservés à des caractères spéciaux définis par l'utilisateur
Compatibilité et usages spéciaux	F900 à FFFD	Caractères à usage spécial et représentation alternative de caractères définis par ailleurs dans la norme Unicode afin de faciliter la compatibilité et la cartographie
Byte order marks	FFFE and FEFF	Ces mots au début d'un fichier peut être pris pour signifier que le fichier est Unicode et, respectivement, dans le format UTF-16 big-endian (UTF-16BE) et UTF-16 little-endian (UTF-16LE)

[6] CJC = Chinois, Japonais, Coréen

4.1.2 Pourquoi Unicode a-t-il été créé ?

Le système de caractères existant a été conçu comme un standard visant à faire fonctionner les afficheurs de texte, les impressions à base de texte et les protocoles de transmission de texte série où l'espace était limité.

Dans le cas des écrans vidéo, le problème a été résolu par une puce électronique appelée contrôleur vidéo qui avait vocation à recevoir des codes et à dessiner le caractère correspondant à l'écran. A cet effet, le programme envoyait deux octets à la fois au contrôleur vidéo. Le premier désignait au contrôleur le caractère à dessiner, et le second fournissait "l'attribut" du même caractère (sa couleur, son fond, un clignotement éventuel, le gras, ou le soulignement).

Étant donné que le code désignant le caractère était contenue dans un octet, le nombre de caractères différents pouvant être affichés était limité. En pratique, les 32 premiers caractères ont été utilisés comme caractères de contrôle (les valeurs décimales 13 et 10 caractérisant respectivement le retour chariot et le saut de ligne en constituent quelques unes), et la valeur 255 a été réservée, de sorte que seules 222 possibilités de caractères étaient véritablement disponibles.

Dans les communications série, les possibilités de codage sont encore plus réduites puisqu'un un bit a été dévolu au contrôle de parité.

Les caractères actuels susceptibles d'être dessinés dans ce contexte ont été regroupés sous le vocable "jeu de caractères". Le jeu de caractères ASCII, dans les 7 premiers bits de l'octet, représentent les caractères romains habituels (a à z, A à Z), les chiffres arabes (0 à 9), et quelques caractères supplémentaires et signes de ponctuation couramment utilisés. IBM a créé un jeu de caractères qui inclut des caractères de dessin de ligne en utilisant le $8^{ème}$ bit.

Dessiner d'autres caractères sur l'écran supposait donc que le contrôleur vidéo puisse basculer sur d'autres jeux de caractères afin d'en tirer un caractère différent lors de l'envoi d'une valeur particulière. Différents jeux de caractères ont donc été créés dans ce but. Mais ici, est apparu le premier problème. Dans la mesure où le texte était élaboré sur une base d'un octet pour un caractère, les langues qui nécessitaient un nombre de caractères disponibles supérieur aux possibilités intrinsèques d'un jeu de caractères mono-octet [par exemple le chinois, le japonais et le coréen (Hangul)] ne pouvaient être représentées que partiellement. C'est la raison pour laquelle des jeux de caractères Double et Multi-octets (DBCS et SBCS) ont été développés. Ceux-ci requièrent des techniques d'analyse et de graphisme spéciales.

Une première difficulté de cette structure découle en partie de l'incohérence dans les jeux de caractères individuels. Par exemple, le jeu de caractères de Windows semble être utilisable par les locuteurs de hongrois, mais il manque en fait quatre caractères (O, O, U et U). En dépit des apparences, les Hongrois doivent donc utiliser la page de code mono-octet Europe centrale iso-8859-2 pour disposer d'un jeu complet (la page de code étant une version dans un jeu de caractères). Las, il lui manque le caractère Õ... On voit donc que, si cela est un problème pour ceux qui utilisent le hongrois, imaginez les difficultés rencontrées par ceux qui utilisent des scripts asiatiques.

Un deuxième difficulté découle du fait que le jeu de caractères par défaut défini dans un ordinateur particulier varie selon ses modalités d'installation, lesquelles ne sont pas les mêmes partout. Sur les machines Windows 9x, une modification ne peut être faite qu'à l'aide du panneau de configuration et ne peut l'être, en tout cas, lors de l'exécution. Cela étant, certaines fonctions d'affichage du système qui sont très utiles au programmeur (notamment MessageBox), ne peuvent afficher qu'avec ce jeu de caractères par défaut. Cela signifie que ces écrans fourniront un affichage erroné si les paramètres régionaux ne sont pas correctement réglés par l'utilisateur.

Dans d'autres types de contrôles, un soin considérable est nécessaire par le programmeur pour veiller à ce que le jeu de caractères correct est utilisé pour la langue d'être représenté.

Comme l'exigence de représentation d'un nombre croissant de langues s'est accrue, on aurait pu s'attendre à ce que l'on se contente d'introduire des jeux de caractères supplémentaires. Mais au lieu de cela, la décision a été prise d'aller dans la voie Unicode. Les avantages sont les suivants :

- Étant donné que chaque caractère peut être représenté dans le jeu de caractères Unicode, deux caractères qui se ressemblent peuvent avoir la même valeur Unicode tout en provenant de langues différentes. Cela permet d'obtenir une représentation plus compacte des caractères et signes de ponctuation japonais, chinois et coréens que si des jeux de caractères individuels avaient été utilisés pour chacune de ces langues.
- Il est beaucoup plus facile d'identifier un caractère particulier en Unicode puisque chaque caractère a une valeur unique. Vous n'avez pas besoin de connaître le jeu de caractères utilisé. Cela facilite beaucoup le transfert des données de caractères internationales entre machines. Toutefois, pour afficher les caractères correctement dans les scripts complexes, vous pourriez bien avoir besoin de connaître la *langue* dans laquelle les caractères sont utilisés, ne serait-ce que pour vous assurer qu'ils sont représentés dans les proportions requises par cette langue.

- Puisque les caractères sont identifiés par leur valeur indépendamment du jeu de caractères, les fonctions d'affichage peuvent dessiner des caractères de différentes langues dans une seule chaîne.
- Des jeux de caractères entiers étaient nécessaires dans chaque langue pour fournir l'*ordre de tri*, c'est-à-dire l'ordre alphabétique des caractères. Dans Unicode cela est géré par le système d'exploitation. On peut obtenir une information détaillée sur ce point en consultant le *National Language Support* ("NLS") avec l'API *GetLocaleInfoEx*. (voir le site MSDN).
- Unicode dispose de beaucoup d'espace pour accueillir de nouveaux caractères. Par exemple de nouvelles pages de code devaient être créées pour permettre de représenter le caractère Euro (€), alors qu'il était facile de l'ajouter à Unicode (valeur 20ACh).

4.1.3 Représentation des caractères non-Romains[7]

Ceci est l'une des choses que vous découvrirez dans la programmation Windows, qui, une fois comprises, vous feront apparaître les autres problèmes comme une aimable promenade. Un contrôle dessine en utilisant un *contexte de périphérique*. Chaque contexte de périphérique comporte un *handle de police* qui lui est assigné (sélectionné dans celui-ci) et est utilisé lors du dessin. Le handle de la police, identifie non seulement la police qui sera utilisée pour le dessin, mais également le *jeu de caractères* qui sera mis en œuvre. Chaque police a un ou plusieurs jeux de caractères. Chaque jeu ne contient qu'un nombre limité de caractères. Ainsi, un contrôle ne peut-il pas dessiner un caractère qui ne soit dans le jeu de caractères de la police sélectionnée dans son contexte de périphérique. À moins que le caractère se trouve d'ailleurs (voir ci-dessous), il produira, en lieu et place, un caractère par défaut, peut-être un trait vertical ou un point d'interrogation.
Donc, l'astuce est de trouver la police et le jeu de caractères corrects pour dessiner les caractères qui doivent l'être.
Il ne saurait être question que toutes les polices et jeux de caractères soient installés sur toutes les machines. Les polices et jeux de caractères qui sont chargés lors de l'installation dépendront de la version de Windows utilisée, que ce soit en anglais ou dans une autre langue, et des applications chargées (lesquelles peuvent disposer de leurs propres polices).
Toutes les machines Windows se voient installer au minimum les polices Courier New, Arial, Times New Roman, Symbole, Wingdings, MS Serif, MS Sans Serif et Tahoma. Selon les versions de Windows, ces polices peuvent gérer au moins les scripts d'Europe centrale et occidentale, l'hébreu, l'arabe, le grec, le turc, le balte, le cyrillique et le vietnamien.
Vous pouvez visualiser les polices chargées sur votre machine en consultant le répertoire Windows\Fonts ou dans la rubrique Polices du Panneau de configuration. Vous trouverez également les polices installées répertoriées dans la base de registres en utilisant l'Éditeur de Registres regedit.exe à :
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts (test effectué sous Windows 8).
Cependant, on ne trouve à ces adresses qu'assez peu d'informations sur les polices et l'on ignore notamment les langues qui sont prises en charge par chaque police. La meilleure façon de se renseigner sur ce point est d'aller dans Options Internet/Général du Panneau de configuration et de cliquer sur le bouton Polices. Puis, dans la liste déroulante de liste des jeux de caractères, sélectionnez la langue que vous souhaitez représenter. Une liste des polices ayant des jeux de caractères capables d'afficher la langue spécifiée apparaît alors. La fenêtre de dialogue de choix de fonte commune aux différentes applications procède de manière semblable, sauf que vous choisissez la police d'abord, puis le script au moyen d'une zone de liste déroulante qui montre quelles langues la police est capable d'afficher.
La capacité d'une police particulière peut différer d'une machine à l'autre. Par exemple, la police Times New Roman fournie avec Windows 98 était limitée aux scripts occidental et grec, alors que sur Windows XP on pouvait également y trouver l'hébreu, l'arabe, le turc, le balte, l'Europe centrale, le cyrillique et le vietnamien.
Si vous disposez de Word, vous pouvez voir les caractères disponibles pour chaque police et jeu de caractères en utilisant "Insérer symbole" puis en cliquant sur le sous-ensemble de la gamme qui vous intéresse.

En tant que développeurs, nous sommes plus intéressés par ce qui peut être fait au moment de l'exécution pour nous assurer que la police correcte est installée. On peut énumérer les polices chargées qui conviennent aux jeux de caractères spécifiés utilisant en utilisant l'API *EnumFontFamiliesEx*. C'est une routine callback capable de rendre compte de la langue et du jeu de caractères de deux manières. Soit en utilisant la valeur de CHARSET (une valeur d'octet fixe comprise entre 0 et 255, vous trouverez au + 17h dans la structure LOGFONT), ou en utilisant le nom d'une langue en mots (rapporté dans la structure de ENUMLOGFONTEX) qui est une option plus flexible. Les langues dans les mots rapportés par *EnumFontFamiliesEx* semblent être ceux qui figurent dans

[7] Le romain (ou écriture romaine) est une fonte de caractères dont les caractères sont droits, par opposition à l'italique où ils sont inclinés vers la droite. L'écriture typographique romaine date des années 1465, lorsque l'imprimerie arrive en Italie. Les Italiens, n'employant pas de caractères gothiques, créèrent des polices de caractères inspirées de leur écriture manuscrite humanistique. Les premiers caractères furent réalisés au monastère de Subiaco, près de Rome, ville dont ils prirent le nom. (source Wikipédia)

la combo-box de Options Internet/Général/Polices qui utilise probablement cette fonction pour remplir son contenu.
Généralement les polices sont capables de représenter seulement une sélection limitée de valeurs dans les plages de caractères Unicode. Aussi, sous Windows 2000/XP, l'API *EnumFontFamiliesEx* est-elle étendue de sorte que vous pouvez également recevoir la *gamme Unicode* d'une police particulière à travers la structure FONTSIGNATURE. L'API *GetUnicodeRanges* peut également être utilisée pour découvrir la gamme Unicode d'une police particulière.

Microsoft signale que l'utilisateur de votre application a également la possibilité de sélectionner la bonne police pour afficher la langue désirée à l'aide du dialogue commun fourni par l'API *ChooseFont*, si nécessaire.
Windows 2000/XP utilise des polices dans certains des moyens astucieux pour maximiser leur polyvalence. Par exemple, si une police donnée ne supporte pas un caractère particulier qui doit pourtant être représenté, Windows regarde les "polices liées" pour voir s'il en existe une qui puisse le faire. Les polices "liées" sont contenues dans la clé de registre :

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontLink\SystemLink.

Celles-ci sont ajoutées à chaque fois qu'un nouveau groupe linguistique est ajouté à la machine par l'utilisateur, en utilisant le Panneau de configuration/ Options régionales.

Lorsque vous avez trouvé la police appropriée aux caractères que vous souhaitez représenter, vous obtenez son handle en utilisant les API *CreateFont* ou *CreateFontIndirect* puis vous sélectionnez la police dans le *device context* prête à être représentée. Si votre application fonctionne sur les plates-formes W9x/ME et XP/2000, il est une pseudo-police très pratique que vous pouvez utiliser : MS Shell Dlg. Vous pouvez spécifier cette police soit dans le fichier de ressources, soit dans la structure LOGFONT et elle assignera automatiquement la police correcte en fonction de la plate-forme. Sur les machines W9x/ME, elle correspond à MS Sans Serif et, sur les machines XP/2000, à Microsoft Sans Serif (ou Tahoma si vous indiquez que vous voulez une police à espacement fixe). Les deux, Tahoma et Microsoft Sans Serif (que l'on trouve sur les machines XP / 2000), prennent en charge les scripts de grec, d'hébreu, d'arabe, de turc, de balte, d'Europe centrale, de cyrillique, de vietnamien et de thaïlandais. Notez que Microsoft Sans Serif n'est pas le même que MS Sans Serif. Ce que MS Shell Dlg détermine peut être trouvé dans :

HKEY_LOCAL_MACHINE \ SOFTWARE\Microsoft\Windows\NT\CurrentVersion\FontSubstitutes.

4.1.4 Que peut-on faire avec les API ANSI ?

Les API ANSI sont toutes disponibles sous Windows 9x et versions ultérieures. En revanche, la plupart des API Unicode (qui affichent des chaînes) ne sont pas disponibles dans Windows 9x et ME. Donc, si vous développez pour ces plates-formes, il est important de connaître les capacités des API ANSI en matière d'affichage des caractères non-romains (italiques, par exemple).
Les API ANSI de représentation de texte acceptent des valeurs de caractères codées sur un octet. Par conséquent, elles ne peuvent représenter des valeurs de caractères Unicode du fait du codage de ces derniers sur 16 bits. L'API affichera selon le jeu de caractères de la police sélectionnée dans le *device context*. La plupart des jeux désignent les mêmes caractères de 0 à 7Fh mais divergent au-delà. Dans Testbug, j'ai pu démontrer comment la sortie de DrawTextA variait avec différents jeux de caractères dans les polices sélectionnées dans le *device context*. Sur une machine Windows 98 de base avec aucune autre police chargée, DrawTextA pourrait représenter des caractères de différents jeux incluant l'hébreu, l'arabe, le grec, le turc et le russe.

Si on pouvait charger plusieurs jeux de caractères dans les polices, les API pourraient représenter plus de caractères. Les *Double Byte Character Sets* sont pris en charge par les API ANSI au moins dans la version extrême-orientale de Windows. Vous pouvez également produire un contrôle *RichEdit* utilisant les API ANSI pour afficher des caractères de toutes polices et jeux de caractères installés sur la machine. Vous indiquez le contrôle que l'on peut utiliser en intégrant dans le texte lui-même (en utilisant des commandes de mise en forme). De *RichEdit* Version 2, (disponible sur Windows 98 et au-dessus, mais en option sur Windows 95), vous pouvez passer la commande pour afficher les caractères Unicode, en utilisant à nouveau les commandes de formatage.

Puisque la représentation de caractères non-romains ANSI implique nécessairement des jeux de caractères, la capacité du système à trouver des polices de remplacement pour dessiner le caractère est très limitée. Vous serez susceptible d'obtenir beaucoup plus de succès en utilisant des caractères Unicode avec des API Unicode.

4.1.5 Que peut-on faire avec les API Unicode ?

Les API Unicode sont conçues pour afficher naturellement du texte Unicode qu'elles reçoivent en format Unicode 16-bit UTF-16. A partir d'une valeur Unicode 16 bits, elles peuvent représenter le caractère correspondant à cette valeur à condition qu'une police appropriée soit chargée sur la machine et soit sélectionnée dans le *device context*. Dans le cas contraire, un caractère par défaut sera représenté, lequel pourra prendre la forme d'une boîte, d'une ligne ou d'un point d'interrogation.

Un petit nombre d'API Unicode est disponible dans Windows 9x et ME et, en particulier, *TextOut* et *ExtTextOut*. Ainsi, même sur cette plate-forme (comme démontré du reste par Testbug) ces API peuvent afficher du texte Unicode. Cependant Windows 9x et ME n'ont pas la même aptitude que Windows 2000/XP à trouver la bonne police et représenter le caractère correct correspondant à la valeur Unicode.
Windows NT/2000 et XP fournissent des versions Unicode d'autres API de chaînes de caractères telles que *DrawText* et les contrôles communs Unicode, les menus et les boîtes de dialogue. Ceux-ci sont capables d'afficher du texte Unicode indépendamment des jeux de caractères et pages de code à condition qu'ils aient la police correcte.

Il y a aussi le jeu d'APIs *Uniscribe* qui sont très sophistiquées et conçues pour analyser les caractères Unicode complexes, leurs composants individuels (*glyphes*) et les afficher correctement.

4.1.6 Élaboration des chaînes à transmettre aux API

En tant que développeur, comment obtenez-vous les chaînes de texte Unicode à donner aux API ? La manière traditionnelle de stocker les chaînes Unicode peut consister, soit à :

- mettre le texte Unicode dans un fichier séparé destiné à être chargé au moment de l'exécution de telle sorte que les chaînes puissent être utilisées directement à partir de données; soit
- mettre le texte Unicode dans un fichier de ressources, soit comme une ressource STRINGTABLE, soit RCDATA, ou à l'intérieur des ressources DIALOG ou MENU ; puis au moment de la compilation des ressources seraient alors liées dans l'exécutable, ou dans des DLL spécialisées en langues qui seraient choisies pour le chargement lors de l'exécution.

Ainsi, en utilisant ces méthodes, il n'y avait pas besoin d'outils de développement pour lire les fichiers Unicode, bien que les compilateurs de ressources aient évidemment à le faire. Et par conception, toutes les ressources sont conservées dans le fichier exécutable au format Unicode de toute façon.
La conservation des chaînes comme une ressource signifie que vous devez récupérer la chaîne à utiliser avec *LoadString* ou *FindResource* suivi de *LoadResource*. *FindResource* peut trouver une ressource de langue spécifique parmi plusieurs avec le même ID. Voici comment *LoadString* serait utilisé dans ce cas :

```
PUSH 100h               ; taille du buffer destiné à recevoir la chaîne
PUSH ADDR BUFFER        ; pointeur du buffer destiné à recevoir la chaîne
PUSH 23                 ; identifiant de la chaîne à obtenir
PUSH [hInst]            ; handle de l'exécutable contenant la ressource
CALL LoadStringW
```

Ce code provoque le placement de la chaîne portant l'ID 23 dans BUFFER. De là, elle peut être affichée à l'écran en utilisant une fenêtre ou une boîte de message comme, par exemple :

```
PUSH 40h                ; information + bouton OK
PUSH ADDR TITLE_TEXT    ; pointeur du texte pour le titre
PUSH ADDR BUFFER        ; pointeur de la chaîne 23 juste retrouvée
PUSH [hWnd]             ; doit être l'enfant de la fenêtre principale
CALL MessageBoxW
```

Notez qu'il existe deux versions de l'API, *LoadStringA* et *LoadStringW*. Dans Windows 9x et ME seule la version ANSI peut être utilisée. Des précautions similaires sont nécessaires lorsque vous utilisez *FindResource* et *LoadResource*.
Ce qui est nouveau dans GoAsm est que vous pouvez entreposer vos chaînes Unicode dans votre script source assembleur et les déclarer dans les données, comme vous le feriez avec n'importe quelle chaîne ANSI. En effet, GoAsm lit scripts source Unicode et inclut des fichiers. Voir la Partie 2 pour plus de détails sur la façon dont les chaînes peuvent être localisées dans des données en utilisant GoAsm et GoRC.

4.2 Utilisation des outils "Go" pour écrire des programmes Unicode

En utilisant les outils "Go", vous pouvez créer un programme Unicode utilisant des caractères non-romains de la même manière que vous concevez un programme ANSI. Le seul outil supplémentaire dont vous ayez besoin est un **éditeur Unicode**. Ensuite, vous pourrez utiliser Unicode dans la **ligne de commande**, les **fichiers de sortie**, les **noms de fichiers**, les **commentaires**, les **labels de code et de données**, les **ID de ressources**, les **mots définis**, les **exports** et les **chaînes**.

4.2.1 Types de fichiers source Unicode lisibles par les outils "Go"

GoAsm et GoRC peuvent lire les fichiers source aux formats Unicode UTF-8 et UTF-16, aussi bien que les fichiers ANSI (texte).
GoLink peut également lire les fichiers batch dans ces formats.

UTF signifie "UCS (Universal Character Set) Transformation Format".

On rappelle que la plupart des caractères mondiaux sont répertoriés dans la norme internationale ISO/CEI 10646 qui définit le jeu universel de caractères (UCS en anglais). Chaque caractère y est identifié par un nom et un numéro uniques. Ce numéro est appelé *point de code* (ou *position de code*). La tâche des normes UTS est d'insérer ce point de code dans un système de codage à la fois pratique et universel.

Dans **les fichiers UTF-16**, le principe de base est que chaque caractère est codé sur une suite de un ou deux mots de 16 bits. Dans le cas d'une représentation sur 2 mots, le format de ces derniers sera le suivant :

- le premier mot aura les 6 bits de poids fort égaux à 110110 et sera donc compris dans l'intervalle [0xD800 ... 0xDBFF] (ici en numération hexadécimale) ; ce mot contiendra dans ses 10 bits de poids faible les 10 bits de poids fort de la différence (représentée sur 20 bits) entre le point de code à stocker et le premier point de code supplémentaire U+10000 ;
- le second mot aura les 6 bits de poids fort égaux à 110111 et sera donc compris dans l'intervalle [0xDC00 ... 0xDFFF] (ici en numération hexadécimale) ; ce mot contiendra dans ses 10 bits de poids faible les 10 bits de poids faible du point de code à stocker.[8]

Dans **les fichiers UTF-8**, les caractères Unicode sont codés sous forme de séquences de un à quatre octets. Les critères permettant de distinguer les différentes longueurs de séquence sont les suivants :

- si la caractère rencontré affiche une valeur allant de 0 à 127, il s'agit d'une séquence mono-caractère et la valeur est alors conforme à la norme ASCII,
- si le caractère rencontré affiche une valeur allant de 128 à 255, la séquence est multi-octets avec une longueur établie selon la valeur hexadécimale du premier caractère de la séquence :
  - C2 à DF : séquence de 2 octets
  - E0 à EF : séquence de 3 octets
  - F1 à F4 : séquence de 4 octets

  Une fois identifiée cette balise, les octets suivants de la séquence affichent obligatoirement une valeur supérieure ou égale à 128. Le point de code du caractère codé est réparti entre les différents octets selon une grille de répartition définie dans la norme UTF-8.

La principale caractéristique d'UTF-8 est donc qu'elle est rétro-compatible avec la norme ASCII, c'est-à-dire que tout caractère ASCII se code en UTF-8 sous forme d'un unique octet, identique au code ASCII.[9]. Il en résulte qu'un fichier UTF-8 en anglais apparaîtra donc presque identique à son équivalent ANSI.

La plupart des fichiers UTF-8 contiennent un indicateur d'ordre des octets[10] dans les trois premiers octets du fichier. Cela les identifie comme étant des fichiers UTF-8. **Cet indicateur est requis par les outils "Go"**.
Les fichiers UTF-16 ont aussi généralement un indicateur d'ordre des octets mais il se réduit à 2 octets. Il existe deux formats de fichiers UTF-16, BE (Big Endian) et LE (Little Endian).[11] **Les outils "Go" fonctionnent uniquement avec les fichiers LE pourvus d'un indicateur d'ordre des octets**.
Il existe d'autres formats de fichiers Unicode, mais les UTF-8 et UTF-16 LE avec un indicateur d'ordre des octets sont les plus largement utilisés. Certains éditeurs Unicode vont même jusqu'à faire référence à des fichiers UTF-16LE avec indicateur d'ordre des octets en les qualifiant tout simplement de fichiers "Unicode".

4.2.2 Élaboration d'un fichier source Unicode avec un éditeur Unicode

Vous pouvez créer un fichier source Unicode en utilisant un éditeur Unicode. Il en existe plusieurs. Certains sont gratuits. Celui que vous utiliserez dépendra de votre langue et de sa facilité d'utilisation en programmation. Il existe quelques bons éditeurs gratuits disponibles. Vous pouvez les trouver par un moteur de recherche, mais vous pouvez également consulter le domaine des ressources sur le site Web Consortium Unicode et également celui du site d'Alan Wood. **UniRed** est un bon éditeur Unicode gratuit disponible depuis le site de Jurij Finkel (allez au signet "Progamaro" de ce site puis UniRed. C'est à ce moment que figurent les indications en anglais). Parmi les gratuits, on trouve également **Yudit**, disponible à partir du site de Gaspar Sinai, **Vim** sur le site Vim et **NoteXPad** de Dream Theater Studio (en chinois, tout de même...). Et enfin, **UniPad** de Sharmahd Computing qui, lui, est payant.

---

[8] le format des mots de 16 bits est emprunté à Wikipédia (entrée : UTF-16)
[9] Ce paragraphe, jusqu'à la présente note est emprunté à Wikipédia (entrée : UTF-8)
[10] L'indicateur d'ordre des octets se nomme BOM (Byte Order Mark) en anglais. Cette technique, qui s'apparente à une signature, repose sur l'hypothèse que la séquence d'octets correspondant à l'indicateur d'ordre des octets en UTF-8 a une très faible probabilité d'occurrence dans un texte encodé dans le codage de caractères ancien. C'est notamment le cas en ISO 8859-1. En effet, en UTF-8, l'indicateur d'ordre des octets est codé par la séquence EF BB BF, qui correspond en ISO 8859-1 au texte « ï»¿ ». (source Wikipédia – entrée Indicateur d'ordre des octets)
[11] L'indicateur d'ordre des octets est FF FE en UTF-16 *Little Endian* et FE FF en UTF-16 *Big Endian*. Les notions *Little* et *Big Endian* recouvrent le mode de représentation des bits dans les octets. En général, nous utilisons le *Little Endian* qui veut que le bit le plus significatif soit à gauche alors que le moins significatif est à droite.

Personnellement, j'ai trouvé **NotePad** très fiable. Il existe en deux versions, l'une en anglais, l'autre en chinois. Recherchez «FR» ou «SC» dans le nom du fichier téléchargé.

Lorsque vous utilisez l'éditeur Unicode pour créer un fichier source, vous devez vous assurer que le fichier est enregistré dans le format correct pour les outils "Go", qui est soit UTF-8 avec indicateur d'ordre des octets (BOM), soit UTF-16 LE 8 avec indicateur d'ordre des octets (parfois appelé tout simplement "Unicode").

4.2.3 La ligne de commande Unicode

Sous Windows NT, 2000 et XP, vous pouvez utiliser des caractères Unicode sur la ligne de commande dans la fenêtre MS-DOS (invite de commande), mais cela ne fonctionne que si vous avez la police de la console sélectionnée sur " Lucida console ". Vous changez la police de la console en cliquant droit sur la barre de titre de la console, et en utilisant «propriétés» et l'onglet "police". Sinon cela peut être fait dynamiquement en apportant des modifications au Registre.

De cette façon, vous pouvez définir un mot (en utilisant le commutateur /d) utilisant des caractères Unicode.

Vous pouvez également utiliser des caractères Unicode pour les noms de vos fichiers d'entrée et de sortie.

Notez que les extensions de noms de fichiers doivent être en caractères Romains si l'extension est importante pour les outils "Go". Par exemple GoRC crée un fichier obj à partir d'un fichier .res. Ici "res" ne peut pas être autrement que sous cette forme. De même, l'extension des fichiers de sortie de GoAsm et de GoRC sera toujours en caractères Romains. Par exemple si vous assemblez le fichier Процесса.asm, il produira un fichier appelé Процесса.obj.

GoLink recherche les fichiers assortis des extensions ".obj" et ".res" pour les fichiers d'entrée, et pourvus des extensions ".dll", ".exe", ".drv" et "ocx" pour les fichiers lors de la recherche des importations. Ces extensions doivent être sous cette forme, mais le nom du fichier lui-même peut utiliser tous les caractères autorisés par le système d'exploitation.

4.2.4 Les fichiers de sortie Unicode

GoAsm produit un fichier de listing si le commutateur /l est spécifié sur la ligne de commande et GoAsm, GoRC et GoLink redirigeront la sortie de leur ligne de commande dans un fichier si le caractère de redirection DOS ">" est utilisé. Dans le cas de GoAsm et GoRC le format du premier fichier source est utilisé pour le fichier de sortie. Celui-ci peut être en effet en ANSI, en Unicode UTF-16LE ou en UTF-8. Dans le cas de GoLink, le format du premier fichier batch Unicode est utilisé pour le fichier de sortie. S'il n'y a pas de fichier batch Unicode, le format dépend du système d'exploitation. Consultez le manuel GoLink pour plus de détails.

4.2.5 Les noms de fichiers Unicode dans votre script source

Vous pouvez fréquemment souhaiter spécifier des noms de fichiers dans votre script source, que ce soit des fichiers de données brutes ou des fichiers de ressources. Dans la mesure où GoAsm et GoRC sont aptes à lire les fichiers Unicode, il est possible de libeller ces noms de fichiers en utilisant des caractères non-Romains dans un script source enregistré dans un format Unicode. Sous Windows NT/2000 et XP cela fonctionne de manière fiable puisque GoAsm peut utiliser les versions Unicode des API de traitement des fichiers. Mais sous Windows 9x et ME, bien que Windows prenne en charge les noms de fichiers en Unicode, seules les versions ANSI des API de traitement de fichiers sont disponibles. Aussi, GoAsm convertit-il le nom Unicode en un nom multi-octets au moment de l'accès au fichier. Ceci est fait en utilisant le jeu de caractères et la page de code du moment (sur la base du "locale"). Mais cela signifie que si un nom de fichier contient un caractère qui n'est pas disponible dans la page de code courante, le fichier ne sera pas reconnu. Cela peut se produire, par exemple, si la page de code a été modifiée depuis le fichier a été constitué.

4.2.6 Les *commentaires* Unicode dans votre script source

Les instructions et opérateurs rencontrés habituellement dans le script source doivent toujours être en anglais. Mais les commentaires peuvent utiliser des caractères non-romains et donc être libellés dans toutes les langues. Sur ce point, on rappelle que GoAsm et GoRC ignorent tout simplement les commentaires. Sur ces deux outils, vous pouvez utiliser, au choix, l'indicateur de commentaire spécifique à l'assembleur (point-virgule) ou les indicateurs de commentaires "C" // (une seule ligne) ou / * *commentaire* * /, ce dernier sur une ou plusieurs lignes.

4.2.6.1 Labels de code et de données dans les scripts source assembleur (fichiers ASM)

CODE et DATA dans le script source assembleur sont les noms donnés respectivement aux objets de fonctions et de données.
Dans les modules utilisés pour élaborer le programme, les labels Unicode peuvent être utilisées librement. Par exemple, supposons que vous ayez une fonction qui a זכוכית: pour label. Il s'agit d'un nom en hébreu. *Incidemment, notez que, bien que le mot doive être lu de droite à gauche, GoAsm attend toujours les deux points à droite du mot*. Par ailleurs, ce label ne peut pas être représenté dans la norme ANSI. Mais, s'il s'avère que votre script

source est en Unicode (format UTF-16 ou UTF-8), il apparaîtra correctement dans votre éditeur Unicode. Et vous pourrez alors appeler cette fonction en utilisant CALL זכוכית à l'intérieur du même module (ie. Le même script source et le même fichier d'objet).
Supposons toutefois que le module qui doit appeler la fonction et que la fonction elle-même soient contenus dans des modules différents. Alors, nous devons avoir la certitude que le label est libellé correctement dans le fichier objet de telle sorte qu'il puisse être appelé en utilisant l'Unicode. GoAsm permet cela car il conserve automatiquement tous les labels Unicode dans le fichier objet au format UTF-8. En fait, c'est le seul moyen pratique de conserver ces labels à l'aide des fichiers objets COFF. En effet, dans les fichiers objets COFF et exécutables COFF tous les labels (symboles) sont conservés entant que chaînes ANSI terminées par un zéro. Pour cette raison, le format UTF-16 ne peut être utilisé parce que les zéros sont autorisés dans une chaîne UTF-16 alors que le format UTF-8 peut l'être pour la raison inverse. *Remarque : pour autant que je sache, il n'y a pas de norme pour libeller les symboles Unicode de non-ressources dans les fichiers objet et exécutable COFF, mais il semble probable que UTF-8 sera le format utilisé*.

4.2.6.2 ID de ressources à partir de scripts de ressources (fichiers RC)

Il est assez simple d'avoir des ID de ressource en Unicode, puisque à la différence des labels de code et de données, ils sont toujours libellés en format UTF-16 dans l'exécutable, de toute façon. Cela signifie que vous avez une ressource avec un ID associé qui utilise des caractères Unicode non-romains. La ressource peut alors être récupérée en utilisant la variante Unicode de l'API *FindResource* (qui est *FindResourceW*) et *LoadResource*. Malheureusement *FindResourceW* n'est pas disponible sous Windows 9x et ME. Cela signifie que vous ne pourrez pas utiliser de caractères Unicode non-romains pour nommer vos ID de ressources lors de l'écriture sur cette plate-forme. Au lieu de cela, vous devrez utiliser des caractères qui sont parfaitement adaptés à la norme ANSI de sorte que, sous Windows 9x ou ME, l'API *FindResourceA* puisse être utilisée en lieu et place.

4.2.7 Export des *fonctions* nommées en Unicode à partir de votre exécutable ou d'une DLL et appel de celles-ci à partir d'autres exécutables

Comme on l'a vu ci-dessus, avec GoAsm, le label de fonction Unicode est libellé dans le fichier objet au format UTF-8. Lorsque le fichier objet (et éventuellement d'autres fichiers objets également) sont transformés en fichiers exécutables (soit un fichier "exe" ou une DLL), si certaines des fonctions sont appelées à être exportées, l'éditeur de liens placera le label de la fonction dans la Table des Noms d'Export (Export Name Table) dans le fichier exécutable. Encore une fois, les noms sont conservés ici en tant que chaînes terminées par un zéro d'où l'emploi du format UTF-8. Maintenant, la signification de ceci est que l'exécutable qui appelle la fonction correspondante *au moment de l'exécution* doit également appeler la fonction en utilisant le format UTF-8. Sinon, le nom ne sera pas reconnu. Cela signifie que le nom de la fonction à appeler doit également figurer dans la Table Hint/Nom du Répertoire d'Import de l'exécutable appelant au format UTF-8. Comme indiqué précédemment, GoAsm garantit que cela se fait en libellant le label dans la table de symboles au format UTF-8. L'éditeur de liens doit respecter ce format et l'utiliser dans les informations du Répertoire d'Import. GoLink fait cela.

4.2.8 *Mots définis* Unicode dans votre script source

GoAsm et GoRC attendent des instructions du script source qu'elles soient libellées en anglais, mais vous pouvez utiliser des macros et des equates pour rendre votre script source plus compréhensible dans votre propre langue si vous le souhaitez. Voici quelques exemples :

```
функция EQU INVOKE
пересылка EQU MOV
ПолучитьХендлМодуля EQU GetModuleHandle
МодальноеДиалоговоеОкно EQU DialogBoxParam
ИнициализацияДиалога EQU WM_INITDIALOG
СообщКоманда EQU WM_COMMAND
```

4.2.9 Comment les labels et ID Unicode apparaissent dans le débogueur

Il se peut que votre débogueur n'affiche pas les noms Unicode pour les ID et les labels (symboles) correctement. GoBug peut le faire.

4.2.10 Méthodes de conversion utilisées par GoAsm

GoAsm utilise son propre algorithme pour convertir le texte Unicode UTF-16 en UTF-8 et réciproquement.
La conversion de texte Unicode en ANSI et vice versa exige de connaître la page de code correcte à utiliser. Ceci, parce que les caractères ANSI au-dessus de 7Fh (jusqu'à 0FFh) varient en fonction de la page de code. Lors de ces conversions, les outils "Go" utilisent tout simplement les API Windows *MultiByteToWideChar* et *WideCharToMultiByte*. Celles-ci sont conçues pour travailler avec la *page de code courante* de la machine. Par défaut (pour la version anglaise de Windows) elles sont positionnées sur la page de code Windows. Mais vous

pouvez modifier cette situation en changeant les paramètres régionaux sur votre machine. Si votre application repose sur l'utilisation d'une page de code particulière, vous devez vous assurer que la machine exécutant les outils "Go" est réglée sur cette page de code.

4.3 Utilisation de chaînes Unicode dans DATA

Nous avons vu comment utiliser les ressources Unicode, et les chaînes qui sont dans un fichier de ressources. Mais aujourd'hui, GoAsm fournit une manière beaucoup plus simple de stocker et d'utiliser vos chaînes Unicode dans les données comme vous le feriez avec des chaînes ordinaires. Ces chaînes vont alors dans le fichier objet et, lorsqu'il est lié, dans l'exécutable dans leur format Unicode original, prêtes à correspondre avec les API Unicode.

4.3.1 Contrôle du format de chaîne

Dans l'absolu, on doit indiquer à GoAsm comment il doit traiter avec les chaînes de votre script source. Par défaut et en l'absence d'indications contraires, GoAsm mettra les chaînes dans le fichier objet soit au format ANSI, soit en Unicode (UTF-16), ce format étant fonction de celui dans lequel lesdites chaînes sont libellées dans le script source. Toujours dans ce cas, GoAsm convertira donc les chaînes d'un script source Unicode UTF-8 en UTF-16. La raison en est la plupart des API Windows fonctionnent avec chaînes UTF-16 plutôt que des UTF-8 (GoRC fonctionne de la même manière et met les chaînes dans le fichier RES en format UTF-16 comme requis par Windows). Vous pouvez considérer que ce comportement n'a rien d'inhabituel et que, dans un script source enregistré au format Unicode (UTF-16 ou UTF-8), vous pouvez envoyer une chaîne Unicode à l'API *MessageBoxW* en utilisant ce code :

```
PUSH 40h                ; information + bouton OK
PUSH ADDR TITLE_TEXT    ; pointeur vers le texte du titre
PUSH ADDR STRING_23     ; pointeur vers la chaîne 23 dans DATA
PUSH [hWnd]             ; sera l'enfant de la fenêtre principale
CALL MessageBoxW        ; appel de la version Unicode de l'API MessageBox
```

MessageBoxW exige que la chaîne Unicode soit au format UTF-16 et également terminée par un zéro, lequel doit être sur 16 bits. Pour s'y conformer (dans un fichier Unicode) vous devez déclarer la chaîne comme suit (dans cet exemple en grec) :

```
STRING_23    DB 'Μπορώ να φάω σπασμένα γυαλιά χωρίς να πάθω τίποτα'
             DW 0                        ; terminateur nul
```

Il est clair que, pour pouvoir utiliser cette méthode par défaut, **il est important que vous ayez connaissance du format de votre script source**.

4.3.2 Utilisation de DUS (déclaration d'une séquence Unicode)

DUS (Declare Unicode Sequence) permet de déclarer une chaîne en même temps que des caractères de contrôle. Voici un exemple d'une chaîne sur deux lignes: -

```
STRING4 DUS "The strings, new line chars",0Dh,0Ah,"and null are in UTF-16",0
```

4.3.3 Modificateurs L', A' et 8'

L'un des moyens d'instruire GoAsm sur la nécessité de placer une chaîne dans le fichier objet dans un format particulier est d'utiliser les modificateurs L ', A' ou 8. Notez que vous pouvez utiliser des guillemets doubles (par exemple. L") au lieu des simples (ce qui vous permet, au passage, d'utiliser l'apostrophe à l'intérieur de doubles guillemets) et vous pouvez même utiliser des guillemets triples (par ex. L"""Hello""") si vous voulez que la chaîne apparaisse dans le fichier objet comme une chaîne entre guillemets.

Les modificateurs agissent de la manière suivante :

Modificateur L' – La chaîne sera toujours mise dans le fichier objet en format Unicode UTF-16

Modificateur A' – La chaîne sera toujours placée dans le fichier objet au format ANSI

Modificateur 8' – La chaîne sera toujours placée dans le fichier objet au format Unicode UTF-8

4.3.4 Modifications utilisant la directive STRINGS

Si vous devez utiliser un grand nombre de modificateurs L"ou A" dans votre script source, vous trouverez peut-être plus simple de les remplacer par une seule directive STRINGS. Celle-ci prescrit à GoAsm de convertir les chaînes au format requis pour le reste du fichier (ou jusqu'à ce qu'une autre directive STRINGS soit rencontrée). Voici un exemple d'utilisation de cette directive :

```
STRINGS UNICODE    ; à partir de ce point, toutes les chaînes sont converties en UTF-16
STRINGS ANSI       ; à partir de ce point, toutes les chaînes sont converties en ANSI
```

La directive STRINGS agit sur toutes les chaînes entre guillemets dans le fichier sauf en ce qui concerne les noms de fichier (par exemple après #include ou INCBIN). Ceci, parce que ces noms peuvent être convertis au format correct en fonction du système, ce qui relève de GoAsm lui-même. Voir Les noms de fichiers Unicode dans votre script source.

Notez que, en tant que directive à GoAsm, STRINGS est reconnue seulement dans le script source et dans les éventuels fichiers d'inclusion d'extension "a". Elle ne sera pas reconnue dans les fichiers d'inclusion non-"a" parce que ces derniers ne sont pas traités comme des scripts source assembleur, mais simplement comme des listes de mot défini (equates, macros et structs).

4.3.5 Déclaration de données utilisant des modificateurs

Voici un exemple de chaîne Unicode terminée par zéro au format UTF-16 (adaptée en tant que paramètre d'API Windows qui traitent des chaînes), faite en utilisant le modificateur L' :

```
STRING_23    DB L'Μπορώ να φάω σπασμένα γυαλιά χωρίς να πάθω τίποτα'
             DW 0            ; null terminator
```

Notez que Unicode nécessite deux octets nuls en sus de la chaîne qui doit être terminée par un zéro (les caractères sont de 16 bits chacun). Donc, au lieu d'utiliser DB, vous pouvez utiliser DW avec un terminateur nul :

```
STRING_28 DW L'Convert me to UTF-16',0
```

Utiliser la directive STRINGS signifie que vous n'avez pas besoin du modificateur L' :

```
STRINGS UNICODE                         ; toutes les chaînes qui suivent seront en UTF-16
STRING_30 DW 'nem lesz tőle bajom',0     ; chaîne UTF-16 terminée par un zéro
```

Même dans un programme Unicode, il peut y avoir des chaînes occasionnelles qui doivent être au format ANSI. Dans ce cas, vous devez utiliser le modificateur A', par exemple :

```
STRING_36 DB A"This string will always be in ANSI"
```

Parfois, vous voudrez peut-être convertir une chaîne au format UTF-8. Pour cela, vous devez utiliser le modificateur 8' comme, par exemple :

```
STRING_40 DB 8"The strings, new line chars",0Dh,0Ah,8"and null are in UTF-8",0
```

Vous pouvez utiliser le modificateur 8' ici avec DB (sans avoir besoin DW ou DWS) parce que les caractères de contrôle en UTF-8 correspondent toujours avec leurs équivalents ANSI.

4.3.6 Mise en pile de chaînes Unicode terminées par un zéro

De la même façon que pour les chaînes ANSI classiques, la très utile instruction GoAsm PUSH *string* peut également être utilisée avec des chaînes Unicode comme, par exemple (en utilisant le russe, cette fois) :

```
STRINGS UNICODE
PUSH 40h                                ; information + bouton OK
PUSH 'На берегу пустынных волн'         ; pointeur vers le titre
PUSH 'Стоял он, дум великих полн'       ; pointeur vers le texte
PUSH [hWnd]                             ; enfant de la fenêtre principale
CALL MessageBoxW
```

Vous pouvez également pousser en pile des pointeurs de chaînes Unicode en utilisant INVOKE au lieu de CALL comme dans cet exemple en hongrois qui utilise le modificateur L' :

```
INVOKE MessageBoxW, [hWnd],   L'Meg tudom enni az üveget', \
                              L'nem lesz tőle bajom', \
                              40h
```

4.3.7 Utilisation de la chaîne correcte dans les immédiats entre guillemets

Les immédiats entre guillemets travaillent de la même manière que les chaînes entre guillemets ordinaires. Par exemple :

```
STRINGS UNICODE
CMP AX,'л'
```

comparera AX avec la valeur UTF-16 correspondant caractère 'л' cyrillique qui est 43Bh.

Cependant, si l'on introduit le modificateur 8' comme ci-dessous :

```
CMP AX,8'л'
```

l'opérande prendra la valeur 0BBD0h (premier octet 0D0h, deuxième 0BBh) qui est la valeur UTF-8 correspondant à 'л'.

Et comme les chaînes entre guillemets ordinaires, un immédiat entre guillemets dans un script source Unicode enregistré au format UTF-8 sera converti en UTF-16 en l'absence d'utilisation de la directive STRINGS ou d'un quelconque modificateur.

4.3.8 Utilisation des chaînes Unicode dans les structures

Les modificateurs agissent de la même manière dans les structures. Par exemple :

```
MyStruct STRUCT
   DB 0
   DW L"I am a null terminated Unicode string",0
   DB 0
ENDS
;
Label64 MyStruct     ; applique le modèle de structure
```

Indépendamment de la présence éventuelle de la directive STRINGS, la chaîne est en Unicode format UTF-16 en raison du modificateur L". Mais si ce modificateur n'était pas présent, l'état de la directive STRINGS agirait alors au moment où le modèle de structure est effectivement *appliqué*, ce qui est le cas à Label64.

Si l'initialisation de la structure est elle-même modifiée, cette modification prend la priorité :

```
UP STRUCT
   DB L'I would prefer to forget this'
ENDS
DontGive UP <8"I won't let you">
```

Ici, la chaîne est au format Unicode UTF-8, parce que, bien que la structure contienne le modificateur L', ce dernier est lui-même remplacé par le modificateur 8".

En outre, dans la mesure où la chaîne UTF-8 est plus courte que la chaîne d'origine, celle-ci se voit rembourrée avec des zéros à concurrence du nombre d'octets d'écart. Si la chaîne UTF-8 avait été plus longue que la chaîne d'origine, elle aurait été tronquée.

4.3.9 Quand s'applique le modificateur ?

La directive STRINGS s'applique aux macros, equates et structures lorsqu'elles sont *appliquées* dans le script source, et non quand elles sont déclarés. Les chaînes en DATA sont différentes parce qu'elles sont appliquées et établies lorsqu'elles sont déclarées. Ainsi, par exemple :

```
STRINGS ANSI
MyString1="I could be either"    ; une equate
MyString2 DB "I couldn't"        ; une déclaration de donnée ANSI
STRINGS UNICODE
PUSH MyString1                   ; pousse en pile le pointeur de la chaîne Unicode
PUSH ADDR MyString2              ; utilise la chaîne ANSI
```

Mais l'action du modificateur primera toujours sur la directive STRINGS éventuellement présente en amont :

```
STRINGS ANSI
MyString1=A"I know what I am"    ; une equate avec chaîne ANSI
MyString2 DB L"So do I"          ; une déclaration de donnée Unicode
STRINGS UNICODE
PUSH MyString1                   ; pousse en pile le pointeur de la chaîne ANSI
PUSH ADDR MyString2              ; utilise la chaîne Unicode
```

4.3.10 Utilisation de SIZEOF sur les chaînes Unicode

La directive sous la forme SIZEOF *label* renvoie le nombre d'octets à partir du label spécifié jusqu'au label suivant (ou jusqu'à la fin de section). Lorsque vous travaillez avec des chaînes Unicode, il fonctionne de la même manière mais **ne** retourne **pas** le nombre de caractères. Ainsi, par exemple :

```
MyString DW 'Здравствуй Мир (Hello World in Russian)',0
MyStringSize DD SIZEOF MyString
```

Dans MyStringSize vous trouverez la valeur 80 si la chaîne est en Unicode UTF-16, ce qui correspond à 39 caractères exprimés en mots de 16 bits pour la chaîne elle-même (soit 78 octets), plus un nul de 16 bits.

Si vous utilisez des structures ou des equates avec des chaînes qui changent de taille du seul fait que vous produisiez une version Unicode ou ANSI de votre programme à l'aide d'un commutateur Unicode/ANSI, assurez-vous que ce commutateur est déjà défini lorsque SIZEOF est utilisé. En effet, la taille en octets de la structure,

d'un de ses membres, ou de la déclaration de données est mesurée au moment où SIZEOF est utilisé, plus qu'à tout autre moment.

4.4 Commutation Unicode/ANSI

4.4.1 La commutation Unicode/ANSI et son intérêt

Bien que les applications Unicode soient susceptibles de devenir la norme, il est non moins probable que les applications ANSI continueront à fleurir. L'une des raisons est que les applications utilisant beaucoup de texte en caractères latins seront beaucoup plus compactes que leurs homologues en Unicode. À moins qu'elles ne soient soumises à des impératifs critiques de rapidité d'exécution, elles ne doivent pas être écrites en Unicode, car elles ne nécessitent pas le jeu de caractères complet.

Une autre raison, moins pertinente aujourd'hui, est que Windows 95, 98 et ME ne contenaient pas la plupart des API Unicode. Cela signifiait qu'une application utilisant les API Unicode ne fonctionnait pas sur ces plates-formes. A l'inverse, une application qui utilisait uniquement les API ANSI fonctionnait non seulement sur Windows 95, 98 et ME, mais aussi sur Windows NT/2000 et XP. Pour cette raison, de nombreuses applications sont écrites uniquement en versions ANSI de sorte que seule une version de travail sur toutes les plates-formes Windows est nécessaire. Cela fonctionne bien pour les applications qui n'ont pas besoin d'utiliser des caractères non-romains.

Cependant, quand un programme ANSI fonctionne sur NT/2000 et XP, il convient de souligner que le système doit convertir la chaîne d'entrée d'API en Unicode, car il utilise sa version Unicode, et doit faire de même avec la chaîne de sortie de l'API pour la restituer selon la norme ANSI. Cela, non seulement ralentit l'application, mais s'avère, de plus, assez complexe.

4.4.2 Commutation utilisant le chargement à l'exécution

Une façon de tirer le meilleur parti des deux plates-formes Windows 9x/ME et NT/2000/XP avec une seule version de votre application est d'utiliser le *chargement à l'exécution*. Une telle application aurait les caractéristiques suivantes :

- Au démarrage, l'application identifie la plate-forme sur laquelle elle est exécutée en appelant l'API *GetVersionExA* (notez que *GetVersionExW* ne doit pas être appelée car elle est pas implémentée dans Windows 9x/ME).
- L'application n'appelle directement aucune des API qui sont indisponibles sous Windows 9x/ME. La raison en est que quand une application est chargée par le système, le chargeur écrit dans l'application elle-même, telle que chargée en mémoire, les adresses des API utilisées par elles. Si le chargeur de système ne peut pas trouver une API particulière (ou une DLL qui est censée contenir cette API) l'application ne démarre pas du tout.
- Il résulte de ce qui précède, et en cas d'exécution sur NT/2000/XP, que l'application doit obtenir les adresses des API Unicode au moment de l'exécution en utilisant le chargement d'exécution. En fait, il s'agit d'une procédure assez simple et bien établie utilisant les API *LoadLibrary* et *GetProcAddress*. Notez que si vous étiez certain qu'une DLL particulière serait chargée par le système de toute façon parce que vous avez appelé une API directement, préférez alors *GetModuleHandle* à *LoadLibrary* pour cette DLL. Un exemple de ceci est donné dans le programme **Hello Unicode 1** figurant dans ce document. Dans une grande application, la commutation peut être traitée en utilisant carrément des tables et des chaînes dans les données. Un exemple de ceci est donné dans le programme "**Run Time Loading**".
- Au moment de l'exécution, l'application appelle les API ANSI de la manière habituelle en cas d'exécution sous Windows 9x/ME et les API Unicode en utilisant les adresses qu'il a trouvées au démarrage en cas d'exécution sous NT/2000/XP. Notez que toutes les API doivent être traitée de cette façon, beaucoup d'entre elles étant par ailleurs disponibles sur les deux plates-formes.

4.4.3 Commutation utilisant le même script source

Un développeur peut vouloir constituer deux versions d'une même application, l'une en ANSI, l'autre en Unicode. Cela pourrait permettre d'éviter les conversions Unicode/ANSI du système au moment de l'exécution, et de tirer pleinement parti des fonctionnalités ajoutées de NT/2000 et XP, ou permettre différentes versions linguistiques avec des caractères non-romains sur une plate-forme NT/2000 et XP . C'est là où la commutation Unicode/ANSI entre en jeu. En l'utilisant, il devient possible, à partir d'un seul script source, de demander à l'assembleur de produire une version ANSI ou Unicode de l'application au moyen de l'assemblage conditionnel.

Voici les commutations qui sont nécessaires dans un tel script source et dans les éventuels fichiers inclus :

- Commutation des API qui ont les deux versions Unicode et ANSI
- Commutation des constantes

- Commutation des chaînes et immédiates entre guillemets
- Commutation des séquences de chaîne incluant des caractères de contrôle
- Changement de la taille d'une écriture – ou d'une lecture – à partir de l'instruction de mémoire en commutant l'indicateur de type
- Changement de la taille d'un caractère pour mettre les membres de structure et les buffers de données à la taille correcte
- Commutation de l'initialisation des données

4.4.4 Conception de la commutation Unicode/ANSI

Traditionnellement, la commutation entre Unicode et ANSI se fait en définissant le mot UNICODE puis en le testant cela en utilisant les opérateurs d'assemblage conditionnel `#ifdef` ou `#ifndef` ("`if defined`" et "`if not defined`", respectivement). En utilisant cette méthode, il est possible de dire à l'assembleur de s'intéresser uniquement aux parties du script source requises pour la version du programme en cours.

Dans GoAsm vous pouvez définir UNICODE de deux façons : soit en ajoutant `/d` à la ligne de commande, par exemple :

```
GoAsm Myfile /d UNICODE
```

soit en ajoutant cette ligne au début du script source :

```
#define UNICODE
```

Dans les deux cas, GoAsm considère l'indicateur UNICODE comme défini et aussi comme ayant la valeur 1 bien que celle-ci ne lui ait pas formellement été attribuée.

Vous pouvez ensuite utiliser ce commutateur comme ceci :

```
#ifdef UNICODE
     ; assemble ces lignes
     ; si UNICODE est défini
     ; puis saute au #endif
#else
     ; assemble ces lignes
     ; si UNICODE n'est pas défini
#endif
```

Dans une situation typique, vous pourriez établir ces commutateurs :

```
#ifdef UNICODE
     AW=W               ; commutateur pour les API.
     STRINGS UNICODE    ; commutateur pour les chaînes entre guillemets.
     DSS=DUS            ; commutateur pour les séquences de chaînes.
     S=2                ; indicateur de type et commutateur de taille de caractère
#else
     AW=A
     STRINGS ANSI
     DSS=DB
     S=1
#endif
```

Le fait d'utiliser #ifdef signifie que, si vous avez besoin de revenir transitoirement à ANSI au travers de votre script de source, vous pouvez « *dé-définir* » UNICODE au moyen de :

```
#undef UNICODE
```

Il existe d'autres façons d'utiliser l'assemblage conditionnel pour basculer entre les versions Unicode et ANSI. Voir la section **Autres façons de réaliser la commutation**.

4.4.5 Commutation Unicode/ANSI des API

GoAsm gère le commutateur ##AW après le nom d'une API, par exemple :

```
CALL DefWindowProc##AW
```

Le commutateur d'assemblage conditionnel est :

```
#ifdef UNICODE
  AW=W
#else
  AW=A
#endif
```

En utilisant ce code, si UNICODE est défini, alors l'API *DefWindowProcW* est appelée. Sinon, *DefWindowProcA* est appelé. Le double dièse a pour effet de combiner les éléments situés de part et d'autre.

Vous ne devez pas utiliser AW dans le commutateur – vous pouvez utiliser ce que vous voulez, par exemple

```
CALL DefWindowProc##SWITCH
```

Un certain nombre d'API ne possède pas de version A ou W (dans ce cas, une seule version de l'API s'applique aussi bien aux programmes ANSI et Unicode). Avec ces API vous ne pouvez pas utiliser le commutateur ##AW. C'est le cas, par exemple, avec :

```
CALL WriteFile
```

Il existe plusieurs manières d'organiser la commutation et chaque utilisateur a la sienne propre. Voir la section **Autres façons de réaliser la commutation**.

4.4.6 Commutation de constantes

Un exemple typique de constantes devant être commutées réside dans les messages Windows qui ont des versions ANSI et Unicode. Ce qui suit, par exemple, est tiré de la de l'en-tête de fichier Windows CommCrtl.h dans le SDK (WM_USER étant une *constante* de 400h) :

```
#define TB_ADDBUTTONSA          (WM_USER+20)
#define TB_INSERTBUTTONA        (WM_USER+21)
#define TB_INSERTBUTTONW        (WM_USER+67)
#define TB_ADDBUTTONSW          (WM_USER+68)

#ifdef UNICODE
   #define TB_INSERTBUTTON      TB_INSERTBUTTONW
   #define TB_ADDBUTTONS        TB_ADDBUTTONSW
#else
   #define TB_INSERTBUTTON      TB_INSERTBUTTONA
   #define TB_ADDBUTTONS        TB_ADDBUTTONSA
#endif
```

GoAsm peut lire ceci, mais voici une alternative en utilisant la commutation double-dièse :

```
TB_ADDBUTTONSA    = WM_USER+20
TB_INSERTBUTTONA  = WM_USER+21
TB_INSERTBUTTONW  = WM_USER+67
TB_ADDBUTTONSW    = WM_USER+68

TB_INSERTBUTTON   = TB_INSERTBUTTON##AW
TB_ADDBUTTONS     = TB_ADDBUTTONS##AW
```

4.4.7 Commutation des chaînes et immédiates entre guillemets

Ici, nous commutons la directive STRINGS en utilisant :

```
#ifdef UNICODE
   STRINGS UNICODE
#else
   STRINGS ANSI
#endif
```

Si votre script source est toujours sauvegardé en format ANSI (c'est-à-dire en tant que fichier texte ordinaire) vous pouvez omettre la ligne STRINGS ANSI, mais elle sera néanmoins indispensable si vous enregistrez votre script source dans l'un des formats Unicode. Le commutateur vous permet ensuite de passer de chaînes déclarées à des chaînes mises en pile[12] comme ceci :

```
ST36 DB "This string can be in Unicode or ANSI"
INVOKE MessageBox##AW,[hWnd],'Hello switched user',ADDR ST36,40h
```

ou des immediates entre guillemets, par exemple, si le registre est ici également commuté en utilisant SWREG pour être soit AX, soit AL :

```
CMP SWREG,'a'
```

Cela va comparer AX avec la valeur UTF-16 de 'a' si UNICODE, et avec la valeur ANSI de 'a' si ANSI.
En savoir plus sur la directive STRINGS.

[12] On rappelle qu'un PUSH *message* ne met en pile que le pointeur de *message*, le contenu de ce dernier étant déclaré, pour sa part, dans la Section DATA.

4.4.8 Commutation de séquences de chaîne avec des caractères de contrôle

Ici, nous pourrions utiliser ce commutateur :

```
#ifdef UNICODE
  DSS=DUS
#else
  DSS=DB
#endif
```

Vous n'êtes pas obligés d'utiliser DSS pour nommer ce commutateur et utiliser un autre nom.

Maintenant, si vous déclarez cette chaîne dans les données ou dans une structure :

```
MyLabel DSS 'I am a string of varying character',0Dh,0Ah,0
```

Lorsque vous concevez un programme Unicode, DSS se traduit par DUS "déclarer la séquence Unicode", de sorte que la chaîne serait mise dans le fichier objet en format Unicode UTF-16, ainsi que les caractères de fin de ligne et le terminateur NULL sous leur forme 16 bits. Mais, avec un programme ANSI, DSS agit comme DB, de sorte que la chaîne et les caractères de contrôle sont en format 8 bits.

4.4.9 Utilisation de l'indicateur de type S commuté

La lettre S est réservée en tant qu'indicateur de type dans toutes les situations où GoAsm pourrait s'attendre à en trouver un. Donc, vous pouvez avoir le commutateur suivant :

```
#ifdef UNICODE
  S=2
#else
  S=1
#endif
```

S peut être commuté à l'équivalent de l'un des indicateurs prédéfinis de type que sont B, W, D, Q ou T. Dans notre exemple, il est mis soit à W (valeur 2), soit à B (valeur 1). Par conséquent, vous pouvez vous en servir pour contrôler la taille de l'instruction, par exemple :

```
MOV S[EDI],0            ; insère un simple zéro si ANSI, double zéro si Unicode
INC S[COUNT]            ; incrémente l'octet à COUNT si ANSI, le mot si Unicode
LOCAL CharUnder:S       ; constitue un octet local si ANSI, un mot si Unicode
LOCAL BUFFER[256]:S     ; constitue un buffer local de 256 octets si ANSI, 512 si Unicode
```

Vous pouvez préférer libeller le commutateur S sous la forme qui suit qui obtient le même résultat que précédemment mais présente l'avantage d'être plus explicite par sa référence directe aux types B, W, D, Q ou T :

```
#ifdef UNICODE
  S=W
#else
  S=B
#endif
```

4.4.10 Utilisation d'un commutateur de taille de caractère

Le commutateur utilisé pour l'indicateur de type peut également être vu comme un commutateur de taille de caractère. Voici à nouveau l'interrupteur :

```
#ifdef UNICODE
  S=2
#else
  S=1
#endif
```

Et cela vous permet de faire ceci :

```
ADD EDI,S                ; incrémente EDI de 1 si ANSI, de 2 si Unicode
HLabel DB 256*S DUP 0    ; constitue un buffer local de 256 octets si ANSI, 512 si Unicode
```

La même idée peut être utilisée dans des structures formelles comme, par exemple :

```
NMTTDISPINFO STRUCT
  hdr         NMHDR
  lpszText    DD
  szText      DB S*80 DUP 0      ; 80 octets si ANSI, 160 si Unicode
  hinst       DD
  uFlags      DD
  lParam      DD
```

```
NMTTDISPINFO ENDS
```

Vous ne devez pas utiliser S comme commutateur de taille des caractères. Traditionnellement, les mots CHAR ou TCHAR sont utilisés à cet effet et apparaissent dans de nombreux fichiers à inclure pour la commutation.

4.4.11 Autres façons de réaliser la commutation

Vous pouvez préférer rendre la valeur de UNICODE plus explicite, en la définissant précisément comme ayant la valeur 1 :

```
GoAsm Myfile /d UNICODE=1
```

ou

```
#define UNICODE 1
```

ou

```
UNICODE=1
```

ou l'équivalence EQU.

Vous pouvez aussi utiliser ce commutateur :

```
#if UNICODE=1
     ; assemble ces lignes
     ; si UNICODE est "ON"
     ; on va alors à #endif
#else
     ; assemble ces lignes
     ; si UNICODE est "off"
#endif
```

Pour revenir à l'ANSI, vous utiliserez alors :

```
#define UNICODE 0
```

ou

```
UNICODE=0
```

ou l'équivalence EQU.

4.4.11.1 Autres façons de commuter les chaînes

Comme alternative à la commutation de la directive STRINGS, les chaînes peuvent être commutées en utilisant la définition conditionnelle qui suit dans laquelle on met en oeuvre des arguments et des double-dièses (qui font que les éléments de part et d'autre se combinent) :

```
#ifdef UNICODE
   #define TEXT(x) L##x
#else
   #define TEXT(x) x
#endif
```

Il en résulterait alors l'utilisation suivante :

```
MyLabel DB TEXT("The string")
```

4.4.11.2 Autres façons de commuter les API

Il existe plusieurs façons de commuter les API A ou W. Certains modes traditionnels de commutation nécessitent malheureusement les listes des API commutables dans des fichiers inclus. Les méthodes proposées de ce type sont :

```
#ifdef UNICODE
   #define DefWindowProc DefWindowProcW
   #define MessageBox    MessageBoxW
#else
   #define DefWindowProc DefWindowProcA
   #define MessageBox    MessageBoxA
#endif
;
CALL DefWindowProc       ; commutée si UNICODE est défini
CALL MessageBox          ; commutée si UNICODE est défini
```

Ou, vous pouvez utiliser :

```
#ifdef UNICODE
```

```
      AW=W
#else
      AW=A
#endif
;
DefWindowProc = DefWindowProc##AW
MessageBox    = MessageBox##AW
;
CALL DefWindowProc   ; commutée si UNICODE est défini
CALL MessageBox      ; commutée si UNICODE est défini
```

Il existe différentes méthodes de commutation utilisant des macros et des arguments par le biais du double-dièse. Dans l'exemple qui suit, un argument est joint à un non-argument pour veiller à ce que l'appel correct de l'API soit fait (*DefWindowProcA* ou *DefWindowProcW*) selon la version du programme :

```
#ifdef UNICODE
  AW(a)=a##W
#else
  AW(a)=a##A
#endif
;
CALL AW(DefWindowProc)
```

Et voici encore une autre variation sur le même thème :

```
#ifdef UNICODE
  CALLAW(a)=CALL a##W
#else
  CALLAW(a)=CALL a##A
#endif
;
CALLAW (DefWindowProc)
```

Notez qu'un commutateur d'API ne convient qu'aux API qui ont une version ANSI et Unicode distincte. Une liste des API a l'avantage de vous indiquer les API qui ont besoin d'une commutation. Pourtant, le linker va bientôt vous dire si vous essayez à tort d'utiliser une API A ou W qui n'existe pas.

4.4.12 Test de la directive STRINGS

Le fait que des chaînes naturelles entre guillemets (c'est-à-dire sans modificateurs) soient converties ou non en Unicode peut faire l'objet d'un test préalable à l'aide de l'opérateur #if d'assemblage conditionnel :

```
#if STRINGS UNICODE        ; ou #if STRINGS ANSI
```

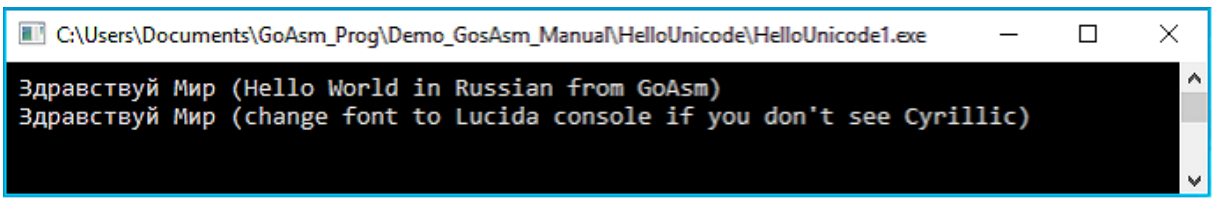
Cette ligne de code teste l'état courant de la directive STRINGS si elle a été déclarée et assemble les lignes suivantes si la condition est vraie (jusqu'au prochain #else, #elseif ou #endif). Notez que si la directive STRINGS n'a pas été déclarée, ce test indique si le script source est au format Unicode ou non (si la directive STRINGS est absente, le format régit ce qui arrive aux chaînes naturelles entre guillemets).

4.5 Programmes de démonstration en Unicode

Les programmes qui suivent sont écrits en Unicode. Si vous souhaitez les reconstituer au moyen d'un simple copier/coller pratiqué sur ce document, vous devez vous équiper d'un éditeur Unicode pour accueillir le script source. En ce qui me concerne, j'ai utilisé NotePad++ qui est multi-langages (dont l'assembleur), gratuit et assez pratique d'utilisation.

4.5.1 Programme HelloUnicode1.asm (mode console)

Nous avons affaire ici à un programme destiné au mode console. Il permet d'obtenir l'affichage suivant (test effectué sous Windows 10) :




```
;-------------------------------------------------------------------------------
```

```
; HelloUnicode1 - Copyright Jeremy Gordon 2004
;
; Ce fichier illustre l'utilisation de chaînes Unicode basées sur:
; 1. Le format naturel du script source
; 2. La directive STRINGS
; 3. Les modificateurs de chaîne A"et L"
;
; Important! Visualisez ce fichier en utilisant Internet Explorer, l'aide ou un éditeur Unicode
; (copiez-le et collez-le dans l'éditeur).
;
; Essayez-le au format "Unicode" qui est UTF-16LE, et essayez à nouveau en UTF-8. Dans les deux
; cas, la sortie de la chaîne Unicode sera en UTF-16 pour convenir à l'API Windows.
;
; Si vous vouliez une chaîne UTF-8, vous pourriez utiliser le modificateur 8"(non utilisé ici).
;
; Assemblage par la ligne de commande : GoAsm HelloUnicode1 (produit le fichier PE COFF)
; Ensuite, édition de liens en tant que programme de console Windows en utilisant GoLink
; comme suit:
; GoLink /console hellounicode1.obj kernel32.dll
; (ajouter -debug coff si vous voulez observer le programme dans le débogueur)
;
; Si vous voyez des points d'interrogation au lieu de caractères cyrilliques (russe) ; dans le
; MS-DOS (invite de commande) dans la fenêtre (console), faites un clic droit sur la barre de
; titre et changez la police en Lucida Console. Cette police est capable de dessiner des
; caractères Unicode sur la console.
; Cette fonctionnalité est uniquement disponible avec NT4, 2000, XP ou plus, ce qui signifie
; que la console sous Windows 9x et ME ne peut pas afficher l'Unicode. Pour cette raison, ce
; programme a effectué une vérification de version pour voir quelle chaîne devait être
; dessinée.
;
; Notez que les API GetStdHandle, GetModuleHandleW, GetVersionExA, lstrlen et WriteFile sont
; toutes localisées dans Kernel32.dll.
;
; L'API WriteConsoleW est nécessaire pour écrire une sortie de texte Unicode sur la console.
, C'est la version Unicode de WriteConsole. Bien que WriteConsoleW soit disponible sur
; Windows 9x (dans Kernel32.dll), elle enveloppe WriteConsoleA (la version ANSI de
; WriteConsole).
;
; Cette démo utilise le "chargement au moment de l'exécution" de WriteConsoleW. C'est l'une des
; façons de faire fonctionner une version de votre programme sur toutes les plateformes (cela
; évite que W9x ne tente de charger des API qu'il ne peut pas trouver). Une autre méthode
; consiste à utiliser les versions ANSI des API (ceci a des implications sur les caractères
; et la vitesse) ou d'utiliser Microsoft Layer for Unicode (ceci peut être implémenté en
; utilisant le commutateur /mslu dans GoLink). Voir le présent chapitre et le chapitre 1
; traitant de GoLink pour plus d'informations.
;-------------------------------------------------------------------------------
; Même en l'absence d'une directive STRINGS ou d'un modificateur L", la chaîne suivante est
; mise dans le fichier objet au format Unicode UTF-16 puisque le script source est dans l'un
; des formats Unicode lus par GoAsm, à savoir, UTF-16LE ou UTF -8 :
; UnicodeString: DB " Здравствуй Мир (change font to Lucida console if you don't see
; Cyrillic)"
;
; Ici, le modificateur A" est utilisé pour s'assurer que la chaîne suivante est placée dans
; le fichier objet au format ANSI malgré le format du script source:
; FUNCTIONERROR_MESS DB A'Sorry - there was an error with this application', 0
;
; mais pour de longues séries de chaînes ANSI, il sera plus facile d'utiliser STRINGS ANSI.
; Chaque chaîne à partir de cette déclaration sera en ANSI en l'absence de modificateur.
; Puisque la directive STRINGS prescrit l'ANSI, la chaîne qui suit est placée dans le fichier
; objet au format ANSI.
; VERSIONERROR_MESS DB 'Sorry - can only write Unicode to console under NT4,2000,XP',0
;
```

```
; En raison de son modificateur L", cette chaîne sera placée dans le fichier objet au format
; Unicode UTF-16 et parce que DW est utilisé au lieu de DB, le retour chariot (0Dh) et le saut
; de ligne (0Ah) seront également rendus correctement au format UTF -16:
; HelloWorldString DW L'Здравствуй Мир (Hello World in Russian from GoAsm)',0Dh,0Ah
;
; Mais, pour de longues séries de chaînes UNICODE, cela sera plus facile à utiliser:
; STRINGS UNICODE
; chaque chaîne, à partir de maintenant, sera en Unicode UTF-16 en l'absence de modificateur.
; On reprend maintenant toutes les déclarations précédentes de manière groupée dans un souci de
; meilleure clarté :
;
;-------------------------------------------------------------------------------------------
DATA
;
UnicodeString DB "Здравствуй Мир (change font to Lucida console if you don't see Cyrillic)"
FUNCTIONERROR_MESS DB A'Sorry - there was an error with this application', 0
STRINGS ANSI
VERSIONERROR_MESS DB 'Sorry - can only write Unicode to console under NT4,2000,XP',0
HelloWorldString DW L'Здравствуй Мир (Hello World in Russian from GoAsm)',0Dh,0Ah
STRINGS UNICODE
;
RCKEEP DD 0     ; variable temporaire
OSVERSIONINFO DD 148
;
size DB 144 DUP 0       ; taille totale de la structure OSVERSIONINFO structure (ANSI version)
        ; = 148 octets
;
;-------------------------------------------------------------------------------------------
CODE
;
START:
PUSH -11D       ; STD_OUTPUT_HANDLE
CALL GetStdHandle       ; récupération, dans EAX, du handle du tampon d'écran actif
MOV EBX,EAX     ; EBX = handle
;**********************
MOV EDI,ADDR OSVERSIONINFO
PUSH EDI
CALL GetVersionExA
MOV ESI,ADDR VERSIONERROR_MESS  ; prêt avec un message en cas de mauvaise plate-forme
CMP D[EDI+10h],1        ; voir si l'ID de la plateforme est NT4,2000, XP et plus
JNA >L4 ; ne pas donner à l'utilisateur le message en ESI

;******* GetModuleHandleW nécessite une chaîne UTF-16 Unicode et actuellement STRINGS prescrit
;******* bien de l'UNICODE, ce qui est OK :
PUSH 'Kernel32.dll'
CALL GetModuleHandleW   ; obtention du handle de Kernel32.dll dans EAX (certainement chargé)
; note: utilisez LoadLibraryW si la DLL n'est pas encore chargée
OR EAX,EAX      ; voir si le handle est bien obtenu
JZ >L2  ; non. Alors, donnez le 2ème message d'erreur

;******* GetProcAddress requiert une chaîne ANSI et utilisez donc le modificateur A"
PUSH A'WriteConsoleW'
PUSH EAX        ; met en pile le handle de Kernel32.dll
CALL GetProcAddress     ; récupération de l'adresse de WriteConsoleW
OR EAX,EAX      ; voir si l'opération a été conduite avec succès
JZ >L2  ; non, on délivre alors un 2nd message d'erreur
MOV EDI,EAX     ; récupère l'adresse de WriteConsoleW dans EDI

;******* WriteConsoleW requiert une chaîne Unicode UTF-16
PUSH 0,ADDR RCKEEP      ; RCKEEP accueille la sortie de l'API
PUSH 53D        ; 53 = nombre de caractères dans la chaîne qui suit
PUSH ADDR HelloWorldString
```

```
PUSH EBX        ; handle du buffer d'écran actif
CALL EDI        ; call WriteConsoleW
OR EAX,EAX      ; voir si l'opération a été conduite avec succès
JZ >L2  ; non, alors on écrit un message d'erreur
;********
PUSH 0,ADDR RCKEEP      ; RCKEEP accueille la sortie de l'API
PUSH 73D        ; 73 = nombre de caractères dans la chaîne qui suit
PUSH ADDR UnicodeString
PUSH EBX        ; handle du buffer d'écran actif
CALL EDI        ; call WriteConsoleW
OR EAX,EAX      ; voir si l'opération a été conduite avec succès
JZ >L2  ; non, alors on écrit un message d'erreur
XOR EAX,EAX     ; retourne zéro
RET     ; retour à Windows

;****** Retours d'erreurs (utilise uniquement des chaînes ANSI) :
L2:
MOV ESI,ADDR FUNCTIONERROR_MESS
L4:
PUSH ESI
CALL lstrlen    ; comptage de la longueur de la chaîne ANSI (résultat dans EAX)
PUSH 0,ADDR RCKEEP      ; RCKEEP accueille la sortie de l'API
PUSH EAX,ESI    ; longueur, adresse de la chaîne
PUSH EBX        ; handle du buffer d'écran actif
CALL WriteFile  ; call WriteFile pour écrire la chaîne ANSI à la console
XOR EAX,EAX     ; retourne zéro
RET     ; retour à Windows
```

4.5.2 Programme HelloUnicode2.asm (GDI)

Voici un programme simple affichant des caractères Unicode dans différentes langues dans une boîte de dialogue modale, puis dans une MessageBox. La boîte de dialogue modale est créée à l'aide de la fonction *DialogBoxIndirectParamW* capable de gérer un modèle intégré sous forme de structures dans la section de données en lieu et place du fichier de ressources externe habituel.

Important! Visualisez ce fichier en utilisant Internet Explorer, l'aide ou un éditeur Unicode (copier et coller dans l'éditeur).

Ce programme ne fonctionne pas sur Windows 9x ou ME car DialogBoxIndirectParamW n'est pas disponible sur ces plates-formes.

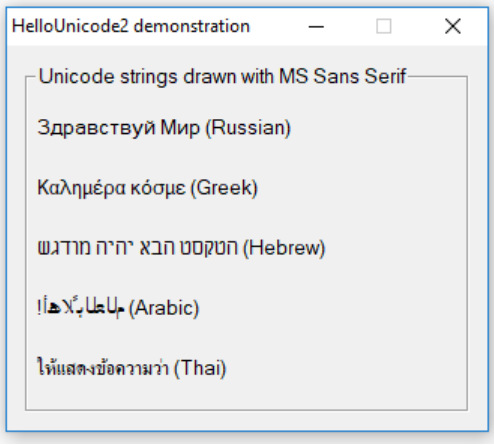


Fig 1 – fenêtre d'accueil

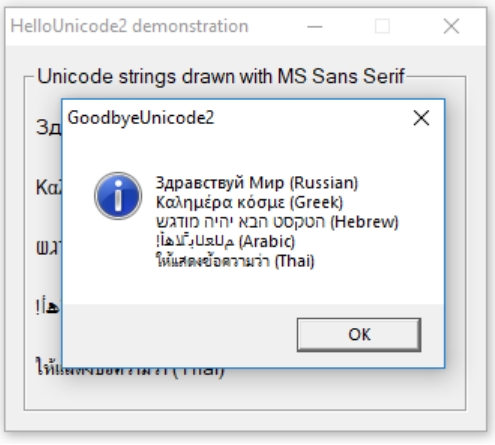


Fig 2 – fenêtre de sortie

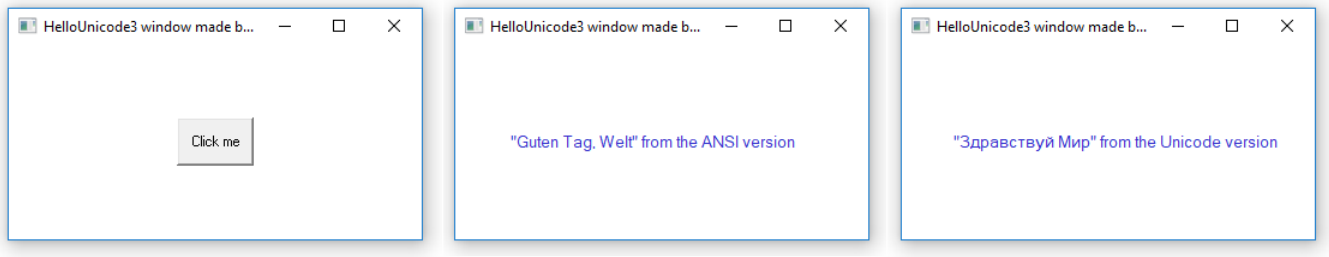
```
;============================================================================================
;----< Visualisez la marque d'ordre des octets UTF-8 dans un éditeur de texte ordinaire
;--------------------------------------------------------------------------------------------
;
; HelloUnicode2 - copyright Jeremy Gordon 2004
;
; Assemblage par la ligne de commande : GoAsm HelloUnicode2
; Cela produit un fichier PE COFF.
```

```
; Puis, procéder à l'édition de liens avec la ligne de commande :
; GoLink HelloUnicode2.obj kernel32.dll user32.dll gdi32.dll
; (ajouter le commutateur /debug coff si vous voulez observer le programme dans le débogueur)
;
;-----------------------------------------------------------------------------------------
DATA SECTION
;
;============== Modèle de dialogue pour la fonction DialogBoxIndirectParamW ==============
ALIGN 4         ; alignement dword impératif
DIALOG_TEMPLATE DD 10000000h |0C00000h |800h |40h | 80h |80000h +20000h
; styles WS_VISIBLE+WS_CAPTION+DS_CENTER+DS_SETFONT+DS_MODALFRAME+WS_SYSMENU+WS_MINIMIZEBOX
        DD 0    ; style étendu
        DW 6    ; nombre d'items dans la boîte de dialogue
        DW 0    ; position x du coin haut gauche
        DW 0    ; position y du coin haut gauche
        DW 160D ; largeur de la boîte de dialogue
        DW 128D ; hauteur de la boîte de dialogue
        DW 0    ; zone de menu (0 = aucun menu)
        DW 0    ; zone de classe (utiliser la classe par défaut)
        DUS 'HelloUnicode2 demonstration',0     ; titre
        DW 10   ; taille en points de la police de caractères
        DUS 'Microsoft Sans Serif',0    ; nous savons par expérience que cette police
                        ; peut afficher les caractères que nous avons
                        ; dans ce fichier. Notez que la chaîne doit être
                        ; spécifiée comme ici car le libellé "MS Sans
                        ; Serif" n'est pas accepté.
;********************************** 1er contrôle dessinant un groupe
ALIGN 4         ; alignement dword impératif
        DD 50000000h +7h        ; (WS_CHILD+WS_VISIBLE)+BS_GROUPBOX
        DD 0    ; style étendu
        DW 6    ; position x du coin haut gauche du contrôle
        DW 6    ; position y du coin haut gauche du contrôle
        DW 148  ; largeur du contrôle
        DW 116  ; hauteur du contrôle
        DW 0    ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
        DW 80h  ; contrôle bouton (valeur atomique)
        DUS 'Unicode strings drawn with MS Sans Serif',0
        DW 0    ; aucun élément à envoyer DlgProc
;********************************** 2ème Contrôle (message en Russe)
ALIGN 4         ; alignement dword impératif
        DD 50000000h    ; (WS_CHILD+WS_VISIBLE)
        DD 0    ; style étendu
        DW 10   ; position x du coin haut gauche du contrôle
        DW 23   ; position y du coin haut gauche du contrôle
        DW 140  ; largeur du contrôle
        DW 10   ; hauteur du contrôle
        DW 1h   ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
        DW 82h  ; contrôle statique (valeur atomique)
        DUS 'Здравствуй Мир (Russian)',0
        DW 0    ; aucun élément à envoyer DlgProc
;********************************** 3ème Contrôle (message en Grec)
ALIGN 4         ; alignement dword impératif
        DD 50000000h    ; (WS_CHILD+WS_VISIBLE)
        DD 0    ; style étendu
        DW 10   ; position x du coin haut gauche du contrôle
        DW 43   ; position y du coin haut gauche du contrôle
        DW 140  ; largeur du contrôle
        DW 10   ; hauteur du contrôle
        DW 2h   ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
```

```
        DW 82h  ; contrôle statique (valeur atomique)
        DUS 'Καλημέρα κόσμε (Greek)',0
        DW 0    ; aucun élément à envoyer DlgProc
;********************************* 4ème Contrôle (message en Hébreu)
ALIGN 4         ; alignement dword impératif
        DD 50000000h    ; (WS_CHILD+WS_VISIBLE)
        DD 0    ; style étendu
        DW 10   ; position x du coin haut gauche du contrôle
        DW 63   ; position y du coin haut gauche du contrôle
        DW 140  ; largeur du contrôle
        DW 10   ; hauteur du contrôle
        DW 3h   ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
        DW 82h  ; contrôle statique (valeur atomique)
        DUS 'מודגש יהיה הבא הטקסט (Hebrew)',0
        DW 0    ; aucun élément à envoyer DlgProc
;********************************* 5ème Contrôle (message en Arabe)
ALIGN 4         ; alignement dword impératif
        DD 50000000h    ; (WS_CHILD+WS_VISIBLE) DD 0
        DD 0    ; style étendu
        DW 10   ; position x du coin haut gauche du contrôle
        DW 83   ; position y du coin haut gauche du contrôle
        DW 140  ; largeur du contrôle
        DW 10   ; hauteur du contrôle
        DW 4h   ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
        DW 82h  ; contrôle statique (valeur atomique)
        DUS 'لاهأ ملعلاب! (Arabic)',0
        DW 0    ; aucun élément à envoyer DlgProc
;********************************* 6ème Contrôle (message en Thaï)
ALIGN 4         ; alignement dword impératif
        DD 50000000h    ; (WS_CHILD+WS_VISIBLE)
        DD 0    ; style étendu
        DW 10   ; position x du coin haut gauche du contrôle
        DW 103  ; position y du coin haut gauche du contrôle
        DW 140  ; largeur du contrôle
        DW 10   ; hauteur du contrôle
        DW 4h   ; identifiant du contrôle
        DW -1   ; l'item suivant décrit la classe
        DW 82h  ; contrôle statique (valeur atomique)
        DUS 'ให้แสดงข้อความว่า (Thai)',0
        DW 0    ; aucun élément à envoyer DlgProc
;------------- Déclaration de tous les messages pour la MessageBox --------
ALIGN 4
VARIOUS_STRINGS DUS 'Здравствуй Мир (Russian)',0Dh,0Ah
        DUS 'Καλημέρα κόσμε (Greek)',0Dh,0Ah
        DUS 'מודגש יהיה הבא הטקסט (Hebrew)',0Dh,0Ah
        DUS 'لاهأ ملعلاب! (Arabic)',0Dh,0Ah
        DUS 'ให้แสดงข้อความว่า (Thai)',0Dh,0Ah,0
;
OSVERSIONINFO               DD 148    ; taille de cette structure simple
                            DB 144 DUP 0      ; et reste de la structure
;*********************************************************************
;* CODE
;*********************************************************************
CODE SECTION
;
;************************************************* PROGRAM START
START:
;************ Avertissons les utilisateurs de Windows 9x de ME des problèmes qui les attendent
MOV ESI,ADDR OSVERSIONINFO    ; structure simple pour GetVersionExA
```

```
PUSH ESI
CALL GetVersionExA            ; la version Unicode de l'API n'est pas disponible sous W9x/ME
CMP D[ESI+10h],1              ; voir à partir de l'ID de la plateforme si NT, 2000, XP et plus
JA >L2                        ; oui, alors on continue
;********************** non, c'est W95/98/ME
PUSH 40h                      ; MessageBox avec icône d'information + bouton OK
PUSH 'HelloUnicode2'
PUSH 'Sorry this program cannot run on your Windows platform'
PUSH 0                        ; faites-en un enfant du bureau
CALL MessageBoxW              ; attente dans la MessageBox tant que OK n'est pas pressé
JMP >L4                       ; et fin
L2:
PUSH 0
CALL GetModuleHandleW         ; on récupère le handle de ce processus dans EAX
;*************************** Maintenant, on crée effectivement la boîte de dialogue
PUSH 0                        ; valeur d'initialisation
PUSH ADDR DlgProc             ; pointeur vers la procédure de dialogue
PUSH 0                        ; cette boîte de dialogue a le bureau comme parent
PUSH ADDR DIALOG_TEMPLATE     ; le modèle est inclus dans le script source
PUSH EAX                      ; handle vers ce processus
CALL DialogBoxIndirectParamW  ; cela ne retourne pas tant que le dialogue n'est pas fermé
L4:
PUSH 0                        ; sortie avec le code zéro
CALL ExitProcess
;
;=====================================================
; PROCÉDURE DE DIALOGUE
;-----------------------------------------------------
DlgProc FRAME hDlg,uMsg,wParam,lParam
USES EBX,EDI,ESI              ; sauvegarde de EBX,EDI et ESI par sécurité
MOV EAX,[uMsg]                ; on récupère le code du message
CMP EAX,110h                  ; est-ce WM_INITDIALOG ?
JZ >.return                   ; on retourne un non-zéro comme requis
CMP EAX,111h                  ; est-ce WM_COMMAND ?
JNZ >.false                   ; non
CMP W[wParam],2h              ; voir si sysmenu a été cliqué
JNZ >.false                   ; non
;********************** Oui, alors on dit au revoir dans un MessageBox
PUSH 40h                      ; MessageBox avec icône d'information + bouton OK
PUSH 'GoodbyeUnicode2'        ; titre
PUSH ADDR VARIOUS_STRINGS     ; texte du corps principal
PUSH [hDlg]                   ; propriétaire (ce dialogue)
CALL MessageBoxW
;*****************************
PUSH 1,[hDlg]
CALL EndDialog                ; fin du dialogue
.false
XOR EAX,EAX                   ; retour d'un zéro comme requis
.return
RET
ENDF
```

4.5.3 Programme TestUnicode3.asm (GDI)




```
;=====================================================================
; HelloUnicode3 - copyright Jeremy Gordon 2004
;=====================================================================
; A éditer en UTF-8 (avec BOM)
;
; PROGRAMME SIMPLIFIÉ GDI WINDOWS DÉMONTRANT
; LES TECHNIQUES DE COMMUTATION UNICODE pour GoAsm
;
; Important! Visualisez ce fichier en utilisant Internet Explorer, l'aide ou un éditeur
; Unicode (faire un copier/coller dans l'éditeur).
;
; Ce fichier est au format UTF-8, mais s'il ne s'agissait pas du "Hello World"
; court en russe (voir BYE_MESS ci-dessous), il pourrait être au format texte
; ordinaire. Il peut produire un exécutable ANSI ou Unicode. Il construit une
; fenêtre avec un bouton intitulé "Click me". Lorsque celui-ci est cliqué, le
; programme informe l'utilisateur du fait qu'il s'agit de la version ANSI ou
; Unicode du programme. Si c'est la version Unicode, il essaie d'écrire la chaîne
; russe courte.
; Ce programme ne fonctionne pas sur Windows 9x ou ME.
;---------------------------------------------------------------------------------------------
; Assemblage par la ligne de commande de 2 manières distinctes :
;              GoAsm HelloUnicode3 => version ANSI.
;              GoAsm /d UNICODE HelloUnicode3 => version Unicode.
; Notez que le mot UNICODE doit être spécifié en majuscules car c'est un mot défini.
; Ceci produit un fichier PE COFF.
; Puis, procéder à l'édition de liens avec la ligne de commande :
; GoLink HelloUnicode3.obj user32.dll kernel32.dll gdi32.dll
; Ajoutez le commutateur /debug coff si vous voulez observer le programme dans le débogueur.
;---------------------------------------------------------------------------------------------
#ifdef UNICODE; si l'on fait une version UNICODE
              STRINGS UNICODE  ; toutes les chaînes entre guillemets doivent être en Unicode
              DSS=DUS          ; DSS signifie toujours DUS (déclare une séquence Unicode)
              AW=W             ; met toutes les API commutées en version Unicode
              S=2              ; type commuté et indicateur de taille
#else         ; en d'autres termes, si ANSI
              STRINGS ANSI     ; toutes les chaînes entre guillemets seront en ANSI
              DSS=DB           ; DSS signifie toujours DB (déclare une séquence d'octets)
              AW=A             ; met toutes les API commutées en version ANSI
              S=1              ; type commuté et indicateur de taille
#endif
;
;---------------------------------------------------------------------------------------------
DATA SECTION
;
hInst         DD 0          ; mémorise le handle du process
CLIENT_RECT   DD 4 DUP 0    ; structure hébergeant le rectangle
PAINTSTRUCT   DD 16 DUP 0   ; structure contenant des éléments de Windows donnés par WM_PAINT
MSG            DD 7 DUP 0   ; structure pour héberger les messages de Windows comme suit :
                            ; hwnd, +4=message, +8=wParam, +C=lParam, +10h=time, +14h/18=pt
```

```
WNDCLASS DD 10D DUP 0       ; structure détenant des données à envoyer à RegisterClass :
                            ; +0  style de classe de fenêtre (CS_)
                            ; +4  pointeur vers la Procédure de Fenêtre
                            ; +8  nb d'octets extra à allouer après la structure
                            ; +C  nb d'octets extra à allouer après l'instance de fenêtre
                            ; +10 handle de l'instance de cette classe de fenêtre
                            ; +14 handle de l'icône de classe
                            ; +18 handle du curseur de classe
                            ; +1C identifie la brosse d'arrière-plan de classe
                            ; +20 pointer du nom de ressource pour le menu de classe
                            ; +24 pointeur de la chaîne du nom de la classe de fenêtre
;
;******************** LOGFONT structure et initialisation des données
LOGFONT STRUCT
              DD 0          ; hauteur
              DB 19 DUP 0   ; attributs qui peuvent être nuls
              DB 0          ; +17h jeu de caractères
              DD 0          ; attributs de précision
              DB S*32 DUP 0 ; +1Ch police de caractères LF_FACESIZE=32 (taille commutée)
ENDS
LF LOGFONT <16,?,204,?,'Microsoft Sans Serif'>
; hauteur = 16, 204 = jeu de caractères Russes, chaîne commutée pour la police
; La spécification du jeu de caractères peut être omise seulement pour la version Windows XP
;
;******************** Table de messages de Fenêtre
; (un vrai programme traiterait beaucoup plus de messages que ce qui est proposé ici)
MESSAGES  DD (ENDOF_MESSAGES-$-4)/8     ; = nb de messages scrutés
          DD 1h,CREATE
          DD 2h,DESTROY
          DD 0Fh,PAINT
          DD 5h,SIZE
          DD 111h,PROCESS_COMMAND
ENDOF_MESSAGES:             ; label utilisé pour déterminer le nombre de messages
;****************************************
; cette chaîne contient un nom unique (qui peut être n'importe quoi) mais doit être terminée
; par NULL - la version ANSI ne nécessite qu'un octet nul. Unicode requiert deux octets nuls,
; donc elle est commutée en utilisant DSS (vous n'avez pas besoin d'utiliser DSS, tout peut
; être utilisé)

WINDOW_CLASSNAME DSS 'WC',0  ; chaîne destinée à contenir le nom de la classe de la fenêtre
;
; Et voici une chaîne disjointe (split). La première partie est commutée en utilisant DSS, la
; seconde en utilisant l'assemblage conditionnel, la troisième partie en utilisant DSS à
; nouveau.
BYE_MESS:
#ifdef UNICODE
              DUS '" Здравствуй Мир" from the Unicode'
#else
              DB ' "Guten Tag, Welt" from the ANSI'
#endif
              DSS ' version'; ce qui suit donne le nombre de caractères (pas le nombre d'octets)
BYE_MESSCHARS DD $-BYE_MESS/S
;
; Cette chaîne et sa terminaison NULL sera toujours en Unicode puisqu'elle est utilisée avec
; MessageBoxW qui est disponible sous Windows NT, 2000, XP et W9x. DUS (Declare Unicode
; Sequence) est un opérateur spécifique à GoAsm qui est très utile lors de la déclaration Unicode
; de séquences dans, par exemple, des modèles de dialogue dans la section de données

SORRYWX_STRING    DUS 'Sorry, your Windows platform cannot use the Unicode'
                  DUS 'versions of the APIs deployed by this program.',0
;
;-------------------------------------------------------------------------------
```

```
ALIGN 4
hButton       DD 0          ; handle du bouton
CLICKMESS     DSS 'Click '  ; À moins que vous n' utilisiez des parenthèses, GoAsm fait des
                            ; opérations arithmétiques de gauche à droite.
BUFFER DB SIZEOF CLICKMESS+3*S DUP 0    ; la taille du buffer dépend du commutateur
; la taille de la structure qui suit convient à GetVersionExA et non à GetVersionExW
; (W pas disponible sur W9x/ME)...
OSVERSIONINFO               DD 148            ; taille de cette structure simple
                            DB 144 DUP 0      ; et le reste de la structure
;-------------------------------------------------------------------------------------------
CODE SECTION
;
INITIALISE_WNDCLASS:      ; préparation de la fenêtre
MOV EBX,ADDR WNDCLASS
MOV EAX,9
L1:
MOV D[EBX+EAX*4],0        ; on remplit avec des zéros
DEC EAX
JNS L1
;***** Ajout d'éléments à la classe de fenêtre pour toutes les fenêtres dans le programme....
MOV EAX,[hInst]           ; EAX = handle du process
MOV [EBX+10h],EAX         ; on en fait la classe de fenêtre
PUSH 32512                ; curseur commun IDC_ARROW
PUSH 0
CALL LoadCursorA          ; on récupère dans EAX, le handle de la flèche du curseur
MOV [EBX+18h],EAX         ; et on le stocke dans WNDCLASS
MOV D[EBX+1Ch],6D         ; on définit la brosse d'arrière-plan à COLOR_WINDOW+1
RET
;
PREPARE_TEXT:             ; on prépare le texte de bouton approprié (ANSI ou Unicode)
MOV EDI,ADDR BUFFER
MOV ESI,ADDR CLICKMESS
MOV ECX,SIZEOF CLICKMESS ; cela restitue la taille de la chaîne telle que commutée
REP MOVSB                 ; met la chaîne "Click me " dans le buffer
;******* Ajoutons le mot "me" puis un NULL manuellement
;******* dans ces séquences, S est soit égal à 1 (octet), soit égal à 2 (mot)
;******* et le caractère est également commuté
MOV S[EDI],'m'
ADD EDI,S
MOV S[EDI],'e'
ADD EDI,S
MOV S[EDI],0              ; terminateur NULL (soit 1 ou 2 octets)
MOV EAX,ADDR BUFFER       ; EAX = adresse du texte
RET
;
;-------------------------------------------------------------------------------------------
; Procédure du message WM_SIZE
;-------------------------------------------------------------------------------------------
SIZE:                             ; ajuste la position du bouton
MOV EAX,[EBP+14h]         ; EAX = lParam
AND EAX,0FFFFh            ; obtention mot faible poids = nouvelle largeur zone client
                                  ; de la fenêtre
MOV EDX,64                ; largeur du bouton désirée
SUB EAX,EDX               ; EAX = largeur de la fenêtre excluant le bouton
SHR EAX,1                 ; EAX = distance de la position-x pour centrer le bouton
;*********** Position-x du bouton maintenant dans EAX, la largeur est dans EDX
MOV EBX,[EBP+14h]         ; EBX = lParam
SHR EBX,16                ; obtention mot fort poids = nouvelle hauteur zone client de la fenêtre
MOV ECX,40                ; hauteur du bouton désirée
SUB EBX,ECX               ; EBX = hauteur de la fenêtre excluant le bouton
SHR EBX,1                 ; EBX = distance de la position-y pour centrer le bouton
;*********** Maintenant, EBX = position-y du bouton, ECX = hauteur du bouton
```

```
PUSH 1,ECX,EDX          ; repeinture, hauteur en pixels, largeur
PUSH EBX,EAX,[hButton]  ; position-y, position-x, handle du bouton
CALL MoveWindow         ; seulement une version de cette API
XOR EAX,EAX             ; retour avec zéro
RET
;
;-------------------------------------------------------------------------------------------
; Procédure du message WM_COMMAND
;-------------------------------------------------------------------------------------------
PROCESS_COMMAND:
MOV EAX,[EBP+14h]       ; récupération du handle de l'élément cliqué
OR EAX,EAX                      ; est-ce un handle de contrôle ?
JZ >.ignore                     ; non, on ne sait pas ce que c'est, alors...
MOV EDX,[EBP+10h]       ; EDX = wParam
SHR EDX,16D             ; on récupère le mot de poids fort et on regarde si c'est BN_CLICKED (0)
JNZ >.ignore                    ; non
CMP EAX,[hButton]       ; est-ce le bouton qui a été cliqué ?
JNZ >.ignore                    ;non, on ne sait pas ce que c'est, alors...
PUSH EAX
CALL DestroyWindow      ; on détruit le bouton - soyons impitoyables !
MOV D[hButton],0        ; on efface même son handle
PUSH 1,0,[EBP+8h]       ; EBP+8h = handle de la fenêtre principale
CALL InvalidateRect     ; et nous devons repeindre toute la fenêtre
.ignore
XOR EAX,EAX             ; retour avec zéro
RET
;
;-------------------------------------------------------------------------------------------
; Procédure du message WM_CREATE
;-------------------------------------------------------------------------------------------
CREATE:                 ; l'un des rares messages traités par ce programme
PUSH 0,[hInst],6D,[EBP+8h]      ;id=6, parent = fenêtre principale (handle dans EBP+8h)
PUSH 0,0                ; taille traitée plus tard (sur WM_SIZE)
PUSH 0,0                ; position traitée plus tard (varie avec WM_SIZE)
PUSH 50000000h          ; (CHILD+VISIBLE)
PUSH 0                  ; texte (s'il est écrit maintenant, il serait dans une mauvaise police)
PUSH 'BUTTON'           ; classe bouton (Chaîne à terminaison nulle au format commuté)
PUSH 0                  ; style de fenêtre étendu
CALL CreateWindowEx##AW ; dessin d'un bouton avec retour du handle dans EAX
MOV [hButton],EAX       ; mémorisation du handle du bouton
;****************** Obtention de la police appropriée à utiliser
PUSH 12D                        ; valeur correspondant à ANSI_VAR_FONT
CALL GetStockObject     ; Obtention du handle de ANSI_VAR_FONT détenu par Windows
PUSH 0,EAX,30h,[hButton] ; 0=pas de re-dessin, police ANSI, WM_SETFONT, handle
CALL SendMessage##AW    ; utilisez-la pour écrire le texte suivant (API commutée)
;******************
CALL PREPARE_TEXT       ; préparation du texte de bouton approprié (ANSI ou Unicode)
PUSH EAX,0,0Ch,[hButton] ; EAX = adresse du texte, 0Ch = WM_SETTEXT
CALL SendMessage##AW    ; ajout du texte du bouton maintenant (API commutée)
XOR EAX,EAX                     ; retour de zéro pour faire une fenêtre
RET
;
;-------------------------------------------------------------------------------------------
; Procédure du message WM_DESTROY
;-------------------------------------------------------------------------------------------
DESTROY:                        ; l'un des rares messages traités par ce programme
PUSH 0
CALL PostQuitMessage    ; sortie via la boucle de message
STC                             ; on va à DefWindowProc aussi grâce à Cf = 1
RET
;
; Cette procédure est appelée par la procédure PAINT
```

```
PAINT_TEXT:

;******************
; A l'entrée, EDI contient le contexte de périphérique
MOV EBX,ADDR CLIENT_RECT
PUSH EBX,[EBP+8h]
CALL GetClientRect       ; obtention de la zone client de la fenêtre dans CLIENT_RECT

;****************** Centrage du texte...
MOV EAX,[EBX+8h]         ; EAX = largeur disponible
SUB EAX,258              ; largeur supposée du texte. En pratique, vous la mesureriez
JNC >L0
XOR EAX,EAX              ; une fenêtre trop mince provoque EAX = 0
L0:
SHR EAX,1                ; on réduit la différence de moitié
MOV ESI,EAX              ; et on donne le résultat à ESI pour position-x
MOV EAX,[EBX+0Ch]        ; EAX = hauteur disponible
SUB EAX,16               ; hauteur du texte
JNC >L1
XOR EAX,EAX              ; une fenêtre trop étroite provoque EAX = 0
L1:
SHR EAX,1                ; on réduit la différence de moitié
MOV EBX,EAX              ; et on donne le résultat à EBX pour position-x
;****************** Définition de la police et de la couleur
PUSH ADDR LF             ; voir l'initialisation de LOGFONT dans les données
CALL CreateFontIndirect##AW       ; obtention du handle de la police dans EAX (API commutée)
;*** Maintenant, quelque chose que vous ne pouvez pas faire en "C"...
PUSH EAX                 ; on pousse en pile le handle de la police maintenant pour
                         ; une utilisation ultérieure par DeleteObject.
;******************
PUSH EAX,EDI             ; handle de la police, contexte de périphérique (DC)
CALL SelectObject        ; sélectionne le handle de police dans DC
PUSH EAX,EDI             ; met en pile la police ancienne, DC pour le SelectObject ultérieur
;******************
PUSH 0CC3333h,EDI        ; couleur, contexte de périphérique
CALL SetTextColor
;******************
PUSH [BYE_MESSCHARS]     ; longueur du message en caractères
PUSH ADDR BYE_MESS               ; adresse de la chaîne
PUSH EBX,ESI,EDI                 ; coordonnée y, coordonnée x, DC
CALL TextOut##AW                 ; écrit la chaîne à l'écran (API commutée)
;******************
CALL SelectObject                ; désélecte la police du DC
CALL DeleteObject                ; et effacement de celle-ci
;******************
RET
;
;---------------------------------------------------------------------------
; Procédure du message WM_PAINT
;---------------------------------------------------------------------------
PAINT:
PUSH ADDR PAINTSTRUCT,[EBP+8h]     ;EBP+8h = handle de la fenêtre
CALL BeginPaint          ; obtention du contexte de périphérique à utiliser, initialise paint
CMP D[hButton],0         ; voir si le bouton est toujours là
JNZ >.donothing          ; oui, ne rien faire qui enlève WM_PAINT de la file d'attente
MOV EDI,EAX              ; met le contexte de périphérique dans EDI pour l'instruction qui suit
CALL PAINT_TEXT
.donothing
PUSH ADDR PAINTSTRUCT,[EBP+8h]     ;EBP+8h = handle de la fenêtre
CALL EndPaint
XOR EAX,EAX              ; ne pas aller à DefWindowProc en plus
RET
```

```
;
; ********** Ceci est une procédure générale de fenêtre qui, dans un
;********** programme ordinaire, traite tous les messages envoyés à la fenêtre
GENERAL_WNDPROC:          ; EAX peut être utilisé pour transmettre des informations au CALL
PUSH EBP                  ; utilisez EBP pour éviter d'utiliser EAX qui peut contenir des informa-
tions
MOV EBP,[ESP+10h]         ; EBP = uMsg
MOV ECX,[EDX]             ; récupération du nombre de messages à traiter
ADD EDX,4                 ; sauter par-dessus le dword contenant la taille de la table des message
L2:
DEC ECX
JS >L3
CMP [EDX+ECX*8],EBP        ; est-ce le message approprié dans la table ?
JNZ L2                    ; non
MOV EBP,ESP
PUSH ESP,EBX,EDI,ESI      ; sauvegarde des registres comme requis par Windows
ADD EBP,4                          ; réajustement de pile pour pointer les paramètres du WndProc
                                   ; (WndProcTable) car nous sommes dans un sous-programme
; Maintenant [EBP+8]=hwnd, [EBP+0Ch]=uMsg, [EBP+10h]=wParam, [EBP+14h]=lParam,
CALL [EDX+ECX*8+4]        ; appel de la procédure correspondant au message
POP ESI,EDI,EBX,ESP
JNC >L4                   ; nc= valeur de retour dans EAX => ne pas appeler DefWindowProc
L3:
PUSH [ESP+18h],[ESP+18h],[ESP+18h],[ESP+18h]    ; empilage des paramètres d'appel de
                                        ; WndProcTable (notez que ESP se décrémente de
                                        ; 4 unités à chaque PUSH)
CALL DefWindowProc##AW    ; (API commutée)
L4:
POP EBP
RET
;
;=============================================================================
;                Procédure de fenêtre appelée par Windows
;-----------------------------------------------------------------------------
WndProcTable:
MOV EDX,ADDR MESSAGES     ; EDX pointe la liste de messages à traiter
CALL GENERAL_WNDPROC      ; appel du gestionnaire de messages générique
RET 10h                            ; restitue la pile comme requis par l'appelant
;
;=============================================================================
;                Point d'entrée du programme
;-----------------------------------------------------------------------------
START:
#ifdef UNICODE            ; assemble ceci si c'est la version Unicode (jusqu'à #endif)
MOV ESI,ADDR OSVERSIONINFO         ; ESI pointe la structure pour GetVersionExA
PUSH ESI
CALL GetVersionExA        ; version ANSI de l'API (fonctionne sur toutes les plateformes)
CMP D[ESI+10h],1          ; voir à partir de l'ID de la plateforme si c'est NT, 2000, XP ou plus
JA >L6                    ; oui, c'est la version Unicode sur NT, 2000, XP, alors on continue
;********************** Cas où c'est la version Unicode sur W95/98/ME...
PUSH 40h                  ; MessageBox avec icône d'information + bouton ok
PUSH 'HelloUnicode3'      ; la chaîne est en Unicode à cause de STRINGS UNICODE
PUSH ADDR SORRYWX_STRING
PUSH 0                    ; la MessageBox est l'enfant du bureau
CALL MessageBoxW          ; affichage de la MessageBox jusqu'à ce que OK soit pressé
PUSH 0                    ; code de sortie = FALSE (notification d'échec)
CALL ExitProcess          ; retour à Windows
L6:
#endif
;********************** Cas d'une version ANSI ou Unicode sur NT, 2000 ou XP
PUSH 0
CALL GetModuleHandle##AW  ; récupération du handle du process (API commutée)
```

```
MOV [hInst],EAX                    ; mémorisation du handle
CALL INITIALISE_WNDCLASS ; préparation structure classe de fenêtre, WNDCLASS = EBX
;********** Enregistrement de la classe de fenêtre à utiliser par le programme
MOV D[EBX],1h+2h+40h     ;CS_VREDRAW+CS_HREDRAW+CS_CLASSDC (style de classe de fenêtre)
MOV D[EBX+4],ADDR WndProcTable     ; procédure de fenêtre
MOV D[EBX+24h],ADDR WINDOW_CLASSNAME    ; nom de classe de fenêtre
PUSH EBX                 ; adresse de la structure avec les données de la classe de fenêtre
CALL RegisterClass##AW   ; enregistrement de la classe de la fenêtre (API commutée)
;********** Affichage de la fenêtre
PUSH 0,[hInst],0,0       ; propriétaire = bureau
PUSH 200D                ; hauteur
PUSH 360D                ; largeur
PUSH 50D,50D             ; position y puis x
;***** définition du style de la fenêtre
PUSH 90000000h +0C00000h+40000h +80000h +20000h +10000h +2000000h
;   (POPUP+VISIBLE)+CAPTION+SIZEBOX+SYSMENU+MINIMIZEBOX+MAXIMIZEBOX+CLIPCHILDREN
; le titre de la fenêtre envoyé ici est modifié - en utilisant la directive STRINGS
PUSH 'HelloUnicode3 window made by GoAsm'       ; titre de la fenêtre (format commuté)
PUSH ADDR WINDOW_CLASSNAME         ; nom de classe de fenêtre (commuté)
PUSH 0                   ; style de fenêtre étendu
CALL CreateWindowEx##AW  ; construit la fenêtre, retourne le handle dans EAX (API commutée)
;********************** Boucle principale de messages
.messageloop
PUSH 0,0,0
PUSH ADDR MSG
CALL GetMessage##AW       ; attente du message de Windows (API commutée)
OR EAX,EAX               ; est-ce WM_QUIT ?
JZ >.quit                ; oui
PUSH ADDR MSG
CALL TranslateMessage    ; non, donc convertit le message en caractère si nécessaire
PUSH ADDR MSG
CALL DispatchMessage##AW ; envoie le message à la procédure de fenêtre (API commutée)
JMP .messageloop         ; une fois le message traité, retour en arrière pour en attendre un autre
.quit                    ; correspond au message WM_QUIT
PUSH [hInst],ADDR WINDOW_CLASSNAME
CALL UnregisterClass##AW ; on s'assure que la classe est supprimée (API commutée)
PUSH [MSG+8h]            ; code de sortie (envoi du contenu de wParam)
CALL ExitProcess         ; retour à Windows de la manière appropriée
```

4.5.4 Programme RunTimeLoading.asm (GDI)

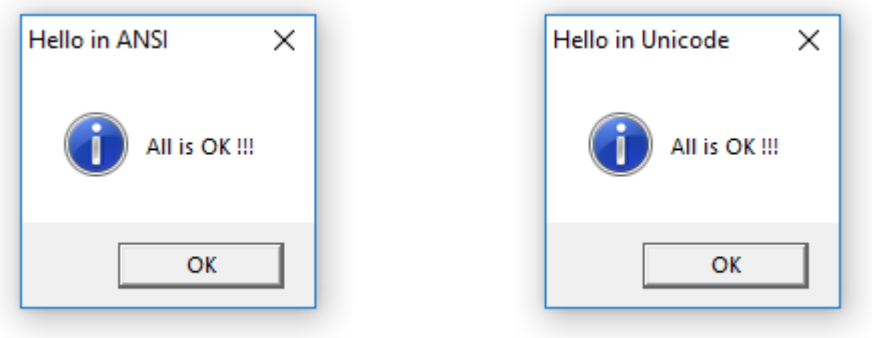
Ce code montre comment utiliser les appels commutés aux API pour écrire une application qui fonctionnera non seulement sous Windows 9x/ME, mais également sous NT/2000/XP. Il y a deux raisons d'utiliser les appels commutés ici. Premièrement, une application qui appelle une API qui n'existe pas dans le système ne se chargera pas. Donc, ici, vous vous assurez que lorsque vous utilisez Windows 9x/ME, les appels aux API Unicode ne sont pas directement appelés. Deuxièmement, c'est un moyen facile d'appeler l'API correcte au moment de l'exécution.

L'utilisation de la commutation de cette manière n'est nécessaire que si vous souhaitez utiliser des caractères non latins dans l'application.

Si vous utilisez des caractères latins – par exemple l'anglais – vous pouvez utiliser les API ANSI (elles fonctionnent aussi bien sur NT/2000/XP) ou vous pouvez utiliser Microsoft Layer for Unicode (MSLU). Ceci peut facilement être réalisé en utilisant GoLink, qui a son propre chargeur MSLU. Voir mon article "Writing Unicode Programs" l'un des tutoriels sur le site Web de GoAsm.

Une troisième alternative consiste à utiliser différentes versions de votre programme, une pour 9x/ME, l'autre pour NT/2000/XP.

Selon la version de Windows présente sur votre PC, ce programme affiche l'une ou l'autre des fenêtres ci-dessous :




```
;=====================================================================
; Copyright Jeremy Gordon 27 Octobre 2004
;
; Ce code est au format GoAsm. Voir http://www.GoDevTool.com
;
DATA
;
; Ici, nous nous concentrons uniquement sur les API dans User32.dll.
; Cela doit être fait pour chacune des DLL que nous utiliserons.
;
USERNAME_ASSTRINGS   DB 'GetClassLongA',0          ; USERSCALLS+0h
                     DB 'GetWindowLongA',0         ; USERSCALLS+4h
                     DB 'SendMessageA',0           ; USERSCALLS+8h
                     DB 'RegisterClassA',0         ; USERSCALLS+Ch
                     DB 'SendDlgItemMessageA',0    ; USERSCALLS+10h
                     DB 'CallWindowProcA',0        ; USERSCALLS+14h
                     DB 'CreateWindowExA',0        ; USERSCALLS+18h
                     DB 'UnregisterClassA',0       ; USERSCALLS+1Ch
                     DB 'SetWindowTextA',0         ; USERSCALLS+20h
                     DB 'DefWindowProcA',0         ; USERSCALLS+24h
                     DB 'CreateDialogParamA',0     ; USERSCALLS+28h
                     DB 'GetClassInfoA',0          ; USERSCALLS+2Ch
                     DB 'DialogBoxParamA',0        ; USERSCALLS+30h
                     DB 'MessageBoxA',0            ; USERSCALLS+34h
                     DB 'LoadMenuA',0              ; USERSCALLS+38h
                     DB 'InsertMenuItemA',0        ; USERSCALLS+3Ch
                     DB 'DrawTextExA',0            ; USERSCALLS+40h
                     DB ' ',0                      ; entrée de substitution
                     DB 0FFh                       ; end stop
USERNAME_WSSTRINGS   DB 'GetClassLongW',0          ; USERSCALLS+0h
                     DB 'GetWindowLongW',0         ; USERSCALLS+4h
                     DB 'SendMessageW',0           ; USERSCALLS+8h
                     DB 'RegisterClassW',0         ; USERSCALLS+Ch
                     DB 'SendDlgItemMessageW',0    ; USERSCALLS+10h
                     DB 'CallWindowProcW',0        ; USERSCALLS+14h
                     DB 'CreateWindowExW',0        ; USERSCALLS+18h
                     DB 'UnregisterClassW',0       ; USERSCALLS+1Ch
                     DB 'SetWindowTextW',0         ; USERSCALLS+20h
                     DB 'DefWindowProcW',0         ; USERSCALLS+24h
                     DB 'CreateDialogParamW',0     ; USERSCALLS+28h
                     DB 'GetClassInfoW',0          ; USERSCALLS+2Ch
                     DB 'DialogBoxParamW',0        ; USERSCALLS+30h
                     DB 'MessageBoxW',0            ; USERSCALLS+34h
                     DB 'LoadMenuW',0              ; USERSCALLS+38h
                     DB 'InsertMenuItemW',0        ; USERSCALLS+3Ch
                     DB 'DrawTextExW',0            ; USERSCALLS+40h
                     DB ' ',0                      ; entrée de substitution
                     DB 0FFh                       ; end stop
ALIGN 4
;
```

```
USERSCALLS          DD 18 DUP 0       ; mémorise les appels aux API dans User32.dll
PLATFORM_ID         DD  0             ; pour mémoriser le système sur lequel nous fonctionnons
BUFFER              DB 1000h DUP ?    ; buffer 4K à usage général
MSG_MSGBOX          DB 'All is OK !!!',0       ; message à afficher dans la MessageBox
;
CODE
;
START:
;
;=============================================================================
;          Identification du système sur lequel nous fonctionnons
;-----------------------------------------------------------------------------
MOV ESI,ADDR BUFFER
MOV D[ESI],148            ; taille de la structure OSVERSIONINFO
PUSH ESI
CALL GetVersionExA        ; cette API fonctionne sur les deux plates-formes
MOV EAX,[ESI+10h]         ; EAX = ID de plate-forme
MOV [PLATFORM_ID],EAX     ; mémorisation de cet ID
;
; Obtention des API de User32.dll en utilisant LoadLibrary. Si la DLL doit être
; chargée maintenant utiliser GetModuleHandle. Si la DLL est déjà chargée, ce qui
; sera le cas si, dans votre source, vous avez appelé une API dans la DLL
; directement de la manière habituelle
;
PUSH 'User32.dll'
CALL LoadLibraryA         ; obtention du handle de User32.dll
OR EAX,EAX                ; ce handle est-il erroné (0 dans ce cas) ?
JZ >L50                   ; sortie car erreur
MOV EBX,EAX               ; EBX = handle
MOV EDI,ADDR USERSCALLS   ; récupération emplacement pour entreposer des adresses API
MOV ESI,ADDR USERNAME_ASSTRINGS    ; emplacement liste des noms d'API (ANSI)
CMP D[PLATFORM_ID],1      ; voir si NT, 2000, XP ou ultérieur
JNA >L18                  ; non. On utilise alors des APIs ANSI
MOV ESI,ADDR USERNAME_WSSTRINGS    ; emplacement liste des noms d'API (Unicode)
L18:
CALL GET_CALLS            ; voir cette procédure plus loin dans ce script
JC >L50                   ; sortie car erreur (à un endroit non montré ici)
;
; Maintenant faites une boîte de message. Notez que si vous écrivez dans cette boîte
; en caractères "romains", vous pourriez tout aussi bien utiliser CALL MessageBoxA
; qui fonctionne à la fois sous 9x et NT/XP. Cependant, si vous voulez écrire dans la
; boîte de message en caractères non-romains, par exemple en Chinois. Cyrillique, etc,
; alors vous utiliserez cette version commutée.
;
;---------------- Copie du message à afficher dans la MessageBox ------------
; BUFFER n'étant plus utilisé pour héberger la structure OSVERSIONINFO, il
; va désormais recevoir le message affiché dans la MessageBox. Selon la
; version de Windows, ce message va être constitué en ANSI et en Unicode.
CLD
PUSH EAX
PUSH ESI
PUSH EDI
MOV  ESI,offset MSG_MSGBOX
MOV  EDI,offset BUFFER
XOR  EAX,EAX
L20:
LODSB
CMP D[PLATFORM_ID],1      ; voir si nous sommes sur NT, 2000, XP ou ultérieur
JBE >L254                 ; non => ANSI
STOSW
JMP >L256
L254:
```

```
STOSB
L256:
OR  AL,AL
JNZ L20
;
;------------------------ Affichage de la MessageBox ------------------------
;
PUSH 40h                ; icône d'information + bouton OK dans la MessageBox
CMP D[PLATFORM_ID],1    ; voir si nous sommes sur NT, 2000, XP ou ultérieur
JBE >L244               ; non => ANSI
PUSH L'Hello in Unicode' ; on met en pile le pointeur de la chaîne Unicode avec le modificateur L
JMP >L246
L244:
PUSH 'Hello in ANSI'    ; on met en pile le pointeur de la chaîne ANSI
L246:
PUSH ADDR BUFFER,0      ; on met en pile le message
CALL [USERSCALLS+34h]   ; CALL soit MessageBoxA, soit MessageBoxW
;
;
;                       le reste du code vient ici
;
L50:
PUSH 0                  ; code de sortie (envoi du contenu de wParam)
CALL ExitProcess        ; retour à Windows de la manière appropriée
;
;============================================================================
;            Recherche des adresses des API dans User32.dll
;----------------------------------------------------------------------------
; Le but est de trouver les adresses de toutes les API de User32.dll
; mentionnées dans les listes USERNAME_ASSTRINGS (ANSI) ou USERNAME_WSSTRINGS
; (Unicode) et de les reporter dans la table USERSCALLS.
;
GET_CALLS:
L1:
CMP B[ESI],20h          ; entrée de substitution ?
JZ >L2                  ; oui, alors on saute par-dessus celle-ci
PUSH ESI,EBX
CALL GetProcAddress     ; obtient adresse de l'API dans la DLL
OR EAX,EAX              ; opération réalisée avec succès ?
JZ >L4                  ; non, abandonner
L2:
STOSD                   ; insère l'adresse de l'API dans la destination du CALL
L3:
LODSB                   ; recherché la fin de ce nom d'API
OR AL,AL                ; test effectif de la fin
JNZ L3                  ; non, ce n'est pas la fin
CMP B[ESI],0FFh         ; est-ce la fin de la liste des fonctions ?
JNZ L1                  ; non, alors on utilise cette chaîne
CLC                     ; retour Cf=0 => succès
RET
L4:
STC                     ; retour avec Cf=1 car échec
RET
```

4.6 Résumé de l'implantation d'Unicode dans les outils "Go"

GoAsm	**Formats de fichiers d'entrée pris en charge :** ANSI, Unicode UTF-8 avec BOM et UTF-16LE avec BOM
	Sortie :

Toute sortie à la console sera en Unicode si elle est autorisée par le système ; le fichier de listing et tout fichier résultant d'une redirection de la sortie seront dans le même format que le premier fichier d'entrée.

Unicode peut être utilisé dans :
le nom des fichiers d'inclusion et des fichiers de données brutes;
le nom des fichiers d'entrée et de sortie via la console (ligne de commande);
les mots définis dans la ligne de commande;
les mots définis dans les fichiers source ou d'inclusion (equates, structs et macros);
les commentaires;
les labels de données (permettant l'accès aux données en utilisant Unicode);
les labels de code (permettant des appels aux fonctions en utilisant des noms Unicode, les importations et les exportations vers d'autres exécutables);
les chaînes (voir ci-dessous).

Déclaration des chaînes dans DATA :

DB "..."	- (*action par défaut*) dans un fichier ANSI, aucune conversion; - dans un fichier Unicode, écrire les données en format UTF-16;

et pour modifier l'action par défaut :

STRINGS UNICODE	- les chaînes doivent toujours être au format UTF-16 dans le reste du fichier
STRINGS ANSI	- les chaînes doivent toujours être au format ANSI dans le reste du fichier

et indépendamment de la directive STRINGS :

DUS "....",0Dh,0Ah,0	- déclare une séquence dans DATA en format UTF-16 (chaîne et caractères de contrôle)
DB L"..."	- déclare les chaînes dans DATA en format UTF-16
DW L"..."	- déclare les chaînes dans DATA en format UTF-16
DB 8"..."	- déclare les chaînes dans DATA en format UTF-8
DB A"..."	- déclare les chaînes dans DATA en format ANSI

Mise en pile de pointeurs de chaînes terminées par zéro dans DATA :

PUSH "....."	- (*action par défaut*) dans un fichier ANSI, aucune conversion; - dans un fichier Unicode, mise en pile du pointeur d'une chaîne Unicode terminée par zéro écrite dans DATA au format UTF-16

et pour modifier l'action par défaut :

STRINGS UNICODE	- les mises en pile concernent désormais les chaînes qui seront toutes converties en format UTF-16 sur le reste du fichier
STRINGS ANSI	- les mises en pile concernent désormais les chaînes qui seront toutes converties en format ANSI sur le reste du fichier

et indépendamment de la directive STRINGS :

PUSH L"....."	- mise en pile du pointeur d'une chaîne en UTF-16 terminée par un zéro
PUSH 8"....."	- mise en pile du pointeur d'une chaîne en UTF-8 terminée par un zéro
PUSH A"....."	- mise en pile du pointeur d'une chaîne en ANSI terminée par un zéro

Immédiates entre guillemets :

MOV EAX,'a'	- (*action par défaut*) dans un fichier ANSI, aucune conversion (met dans EAX la valeur du caractère 'a' dans la page de code courante); - dans un fichier Unicode, met la valeur Unicode du caractère 'a' dans EAX

et pour modifier l'action par défaut :

STRINGS UNICODE	- utilise les valeurs de caractère Unicode sur le reste du fichier
STRINGS ANSI	- utilise les valeurs de caractère ANSI sur le reste du fichier

et indépendamment de la directive STRINGS :

MOV EAX,L'a'	- utilise la valeur de caractère Unicode
MOV EAX,8'a'	- utilise la valeur de caractère UTF-8
MOV EAX,A'a'	- utilise la valeur de caractère ANSI

Commutation utilisant l'assemblage conditionnel :
Commutation des API Unicode/ANSI - méthodes variées, facilitées par l'usage du double-dièse
Commutation de la directive STRINGS
Commutation des séquences de caractères Unicode ou ANSI
Commutation du type "S" pour agir de la même manière que les indicateurs de type B,W,D,Q ou T
Commutation de l'indicateur de taille en caractères

GoRC	**Formats de fichiers d'entrée pris en charge :** ANSI, Unicode UTF-8 avec BOM et UTF-16LE avec BOM
	Sortie : Toute sortie à la console sera en Unicode si elle est autorisée par le système ; le fichier résultant d'une redirection de la sortie sera dans le même format que le premier fichier d'entrée.
	Unicode peut être utilisé dans : le nom des fichiers d'inclusion, et des fichiers de données brutes; le nom des fichiers de ressource (c'est-à-dire les icones et bitmaps) le noms des fichiers d'entrée et de sortie via la console (ligne de commande); les mots définis dans la ligne de commande; les mots définis dans les fichiers source ou d'inclusion (equates, structs et macros); les commentaires; les ID de ressource; les types de ressources; les chaînes (voir ci-dessous).
	Chaînes dans la ressource de version, les tables de chaînes, les boîtes de dialogue, les menus, les contrôles et les ressources définies par l'utilisateur : sont toujours converties en format Unicode UTF-16 si elles n'y sont pas déjà, les séquences d'échappement sont autorisées et la conversion de nombre effectuée (*voir le manuel GoRC*)
	Chaînes dans la ressource RCDATA (donnée brute dans un script de ressource) (*action par défaut*) conservé dans le même format que celui du script source, les séquences d'échappement sont autorisées et la conversion de nombre effectuée (*voir le manuel GoRC*) L"....." - la chaîne est convertie au format UTF-16 si elle n'y est pas déjà

GoLink	**Commandes d'entrée :** *Ligne de commande en console* (prompt de commande MS-DOS) : accepte les noms de fichier Unicode s'ils sont supportés par le système d'exploitation. *Fichiers batch* : ils peuvent être en ANSI, Unicode UTF-8 avec BOM et UTF-16LE avec BOM
	Sortie : La sortie à la console sera en Unicode si elle est autorisée par le système; le fichier qui reçoit la redirection de sortie sera dans le même format que le premier fichier batch Unicode. S'il n'y a pas de fichier batch Unicode, le format dépend du système d'exploitation – voir le manuel GoLink.

4.7 Unicode – Informations supplémentaires et liens

The Unicode Consortium *site officiel pour les nouveautés Unicode, mises à jour, informations et liens*

Alan Wood's Unicode resources *un site incontournable, tenu à jour, avec des informations et des ressources*

Dr International *site international de Microsoft pour les développeurs - clair et compréhensible*

SIL (Summer Institute for Liguistics) computer page *outils variés et ressources*

Institute of Estonian language letter database *propose les valeurs Unicode, des jeux de caractères, et recense les exigences des différentes langues*

Newsgroups :

MSDN international newsgroup

Divers articles :

Design a Single Unicode App that Runs on Both Windows 98 and Windows 2000 par F Avery Bishop (Avril 1999).

Supporting Multilanguage Text Layout and Complex Scripts with Windows NT 5.0 par F Avery Bishop, David C Brown et Davis M Meltzer (Novembre 1998).

Get World-Ready - Fonts Global Development and Computing Portal.

Writing Win32 Multilingual User Interface Applications Global Development and Computing Portal.

Configurable Language and Cultural Settings Global Development and Computing Portal.

Character sets par Ken Fowles, Microsoft Typography site.

Unicode Fonts for Windows Computers par Alan Wood.

Autres références :

"The Unicode Standard Version 3.0 publié par The Unicode Consortium.

"Unicode - A Primer" par Tony Graham, M & T Books.

Various articles in the Microsoft Development Library, Knowledge Base and Windows Software Development Kit ("SDK").

"Hello World" par Rob Pike, Ken Thompson AT & T Bell Laboratories Janvier 1992.

"An Essay in Endian Order" 1996 par Dr William T Verts.

"Alternate formats", Février 2000, Network Working Group.

"Under the Hood" (Unicode support in Windows NT) 1998 by Matt Pietrek.

"Forms of Unicode" 1999 by Mark Davis (IBM developerWorks).

Remerciements

Wayne J Radburn, Edgar Hansen, Daniel Fazekas, Leland M George, Dmitry Ilyin, Greg Heller, Rick de Unicode.org, et tous celles et ceux de la communauté **Win32Asm**. Une partie du texte Unicode utilisé dans ce chapitre est due au projet Kermit à l'Université de **Columbia**.