==========

The Flaming Bird Disassembler - (c) Phœnix corp. 1992-94

Fichiers d'extension

==========

Intro

La plupart des utilisateurs d'un désassembleur sont un jour confrontés a un cas particulier qui n'était à l'origine pas prévu par le programmeur, comme par exemple une gigantesque zone de données spécifique au programme à désassembler organisée de manière connue mais qui éxigerait de passer des jours entiers dessus. Pour parer à ce probleme ont été crées les scripts, sorte de mini-langage interprété permettant de gérer la plupart des cas simples, mais vite limités lorsqu'il s'agit de traiter des zones relativement complexes, voire tout un fichier. Il peut par exemple être pratique de pouvoir désassembler systématiquement un certain type de fichier (librairies, datas...) de manière efficace et rapide. C'est dans cette optique qu'ont été crées les fichiers d'extension.

Si vous écrivez des fichiers d'extension que vous jugez d'un certain intérêt (notamment pour le désassemblage de fichiers de type connus), envoyez les moi, ils seront diffusés avec les prochaines versions de TFBD et profiteront a tous les utilisateurs (je vous renverrai la disquette, avec un update de TFBD s'il y a lieu).

==========

Doc

Les fichiers d'extension sont des modules de type TLK ($BC), logés dans le directory 1:Expand. Leur auxtype a une signification particulière: il détèrmine dans quelle mesure l'extension peut s'appliquer au fichier en cours de désassemblage.

bit	15:	'Load File' seulement
bits	8-14:	Réservé (0)
bits	0- 7:	Type de fichier traité (0 = tout type)

Par exemple, une extension d'auxtype $80B3 ne s'appliquera qu'aux applications P16 & GS/OS, $8000 à tout segment de fichier OMF, $0006 aux fichiers binaire et $0000 à tous les fichiers.

Les extensions peuvent etre multi-segments et comporter des resources, mais le champ Entry du segment principal doit pointer sur une structure particulière, definie comme suit:

00	Long	Signature	ASC "TFBD"
04	Word	Version	($0100)
06	Word	Réservé	(doit être mis a 0)
08	Long	TFBD_Call	
0C	... Début du code de l'extension		

Le champ 'TFBD_Call' sera mis en place par TFBD au moment du chargement de l'extension et contiendra un saut long (JMPL) vers un point d'entrée interne à TFBD.

'Version' contient le numéro de version minimum de TFBD nécessaire à la bonne éxécution de l'extention. Pour l'instant, la seule valeur possible est $0100 (v1.0).

La signature est obligatoirement en ascii superieur.

Les extensions sont appelées de TFBD via un JSL, et doivent par consequent se terminer par un RTL.

Attention: une extention n'est pas une commande shell, et ne doit en aucun cas se terminer par un Quit GS/OS.

Pile et page directe:

L'extention est appelée avec, déjà mis en place, sa propre pile et sa propre page directe. La taille totale du buffer est de $800, soit $100 pour la page directe et $700 pour la pile, ce qui devrait être suffisant dans la plupart des cas. Si l'extension a un besoin plus important en pile par exemple, à elle de s'occuper de s'en allouer une autre ou bien de disposer de son propre segment DP/Stack.

Valeurs de la page directe:

Les différentes données passées a l'extension sont stockées dans sa page directe, comme suit:

$80	Long	ObjPtr	Pointeur sur le fichier/segment
$84	Long	ObjLength	Longueur du segment
$88	Word	SegNum	Numéro du segment
$8A	Word	FileType	Type du fichier
$8C	Long	AuxType	Type auxiliaire
$90	Long	SegHead	Pointeur sur le segment header
$94	Long	OMF_Hnd	Handle sur les infos OMF
$98	Word	memID	ID de l'extension

$9A-$AF rffe Réservé

Dans le cas d'un fichier non relogeable, le numéro de segment est a 1, le segment header contient des infos non significatives et il n'y a pas d'infos OMF. memID est spécifique a l'extension, et elle fait ce qu'elle veut avec. Mais comme toute application GS/OS, elle est priée de laisser la mémoire dans l'état où elle l'a trouvée en entrant.

Attention: l'éxécution d'une extention pouvant être suspendue par l'utilisateur, celle-ci ne doit allouer de la mémoire (si besoin est) qu'avec l'ID que TFBD lui a communiqué (et éventuellement ses IDs auxiliaires). Elle ne doit pas s'en créer d'autre via _GetNewID par exemple.

Appels au Flaming Bird Disassembler:

Les appels a TFBD se font à la manière des toolsets: paramètres passés par la pile, numéro de fonction dans X puis JSL. Normalement, un fichier de macros et d'equates est livré avec le soft: si vous ne l'avez pas, gueulez, parceque je ne decrirai pas ici l'organisation du registre X. J'ai les macros, je m'en sers (de même, je me fous royalement des numéros des tools quand je programme !).

Une différence est toutefois à noter entre les appels a TFBD et les appels aux tools: il est inutile de réserver de l'espace dans la pile pour les valeurs de retour.

*------ Offsets & Adresses

La plupart des appels touchant a l'objet nécessitent non pas une adresse mais un offset, c'est à dire l'octet du code que l'on référence, et non l'adresse qu'il occuperait chargé en memoire. Par exemple pour un fichier SYS, l'adresse $2000 est l'offset $0000. Comme bien souvent une extension peut avoir à aller chercher une adresse dans le code (dans une table d'adresses par exemple), il faut auparavant la transformer en offset. Ce problème ne se pose généralement pas avec du code OMF, un fichier OMF ne contenant que des offsets.

Deux appels sont prévus pour ce genre de chose:

- AddrToOffset:

Paramètres d'entrée:	Sortie:
Previous content	Previous content
---------- +4	---------- +4
	Segment number

Address ---------- +2
Offset
---------- <- SP ---------- <- SP

Erreurs: expBadOffset ($0101)

- OffsetToAddr: Appel inverse. Convertit les offsets en adresse, en fonction des ORG du code. Les paramètres sont les inverses de ceux de AddrToOffset.

Erreurs: expBadAddr ($0102)

Pour le problème qui consiste à aller chercher une adresse directement dans le code, par exemple l'adresse referencée par un JMP ou un DA, un appel spécifique est prévu: EvaluateEA, qui évite au programmeur d'extensions de prendre lui-même en compte le segment référencé par l'adresse ou bien la valeur du DBR pour un adressage mémoire absolu, par exemple.

- EvaluateEA: Evaluation d'une adresse effective

Entrée: Sortie:

Previous content Previous content
---------- +8 ---------- <- SP
Segment number
---------- +6
Offset
---------- +4

EARecPtr

---------- <- SP

EARecPtr est un pointeur sur une structure dans laquelle EvaluateEA va inscrire sa réponse; elle est organisée comme suit:

0	Word	Kind
2	Word	Size (en bytes)
4	Word	Shift count
6	Long	Référence

Kind indique de quel type est la référence:
0: Lo Word = Offset, Hi Word = Segment
1: Offset/Segment opérande d'une instruction 816

2: Offset/Segment via un enregistrement OFFSET
3: Adresse absolue (externe au code)
4: Adresse en page directe
5: Indeterminé ("Peut-être" une adresse)

Si TFBD ne trouve aucune adresse à cet endroit (par exemple si l'offset qui lui est passé pointe sur une instruction ou au beau milieu d'une constante), il retourne l'erreur expNoAddr ($0103).

*------ Alignements

Deux appels existent pour se déplacer dans le code sans se soucier du désassemblage:

- PreviousInst: remonte d'une instruction dans le code

Entrée:	Sortie:
Previous content	Previous content
---------- +2	---------- +2
Offset	New offset
---------- <- SP	---------- <- SP

- NextInst: avance d'une instruction

Entrée:	Sortie:
Previous content	Previous content
---------- +2	---------- +2
Offset	New offset
---------- <- SP	---------- <- SP

L'offset est obligatoirement dans le segment courant.

Si l'offset n'est pas aligné sur le début de l'instruction, il est considéré comme tel; “~PreviousInst offset” n'alignera pas l'offset sur le début de l'instruction courante mais sur ce lui de la précédente (c'est le but).

Propriété amusante: un enchainement _PreviousInst - _NextInst aligne l'offset sur l'instruction courante. Mais cette fonction a une commande spéciale:

- Align: aligne l'offset sur l'instruction dans laquelle il pointe.

Entrée: Sortie:

Previous content Previous content
---------- +2 ---------- +2
Offset New offset
---------- <- SP ---------- <- SP

N'a aucun effet si Offset est déjà aligné.

*------ Appels divers

- Message: Affiche un court (max 77 chars) texte sur la ligne du bas.
Le texte doit être une PString (byte + ascii).

Entrée: Sortie:

Previous content Previous content
---------- +4 ---------- <- SP

String pointer

---------- <- SP

- Error: Affiche un message d'erreur ("> ERR: " + Beep + Message), attend une touche puis rend la main a l'extention.
Les paramètres sont les mêmes que pour Message.

*------ Constantes

Les appels suivants servent a définir une constante à un endroit du code. Les paramètres sont identiques à toutes les constantes:

Entrée: Sortie:

Previous content Previous content
---------- +6 ---------- <- SP
Segment number
---------- +4
Offset
---------- +2

Constant size
---------- <- SP

L'appel s'effectue via la macro _AddConst suivi du type de constante (dw, str, ...) en minuscules:

PHW SegNum
PHW Offset
PHW cSize
_AddConst ctype ou ~AddConst ctype;Seg;Off;cSize

Pour les constantes de longueur fixe dw, da, ddb, adrl & adr le paramètre cSize doit être multiple de la longueur de la constante; sinon, il est aligné sur le multiple supérieur le plus proche. Il n'est pris tel quel que pour: db, ds, asc, hex, rev, str & strl. Pour les strings, la longueur comprend l'octet ou le mot de longueur, alors attention sous peine de mauvais désassemblage. Une longueur de zéro peut provoquer des petites surprises également...

*------ Labels

La définition d'un label nécessite 4 paramètres: l'endroit (Seg/Off) où on le place, son champ d'action, son nom et son type (lbl, ent, equ). Son champ d'action représente le nombre d'octets à partir du label qui seront adressés sous la forme "Label+n". En général, c'est la longueur de l'instruction ou de la constante devant laquelle on le place. Un champ d'action de zéro sera automatiquement calculé par TFBD comme indiqué precedament. Appel:

- SetLabel: définition d'un label.

macro: _SetLabel lType ou ~SetLabel lType;Seg;Off;FieldSize;StrPtr
avec lType = lbl, ent ou equ.

Entrée: Sortie:

Previous content Previous content
---------- +10 ---------- <- SP
Segment number
---------- +8
Offset
---------- +6
Field size
---------- +4

Name pointer

---------- <- SP

Tout label précédament défini a l'offset donné sera effacé.

*------ Directives

- AddOrigin(Seg,Off,Org/4): définition d'une origine d'assemblage.

Entrée:

Previous content
---------- +8
Segment
---------- +6
Offset
---------- +4

Origin

---------- <- SP

Sortie:

Previous content
---------- <- SP

- AddMX(Seg,Off,MX): Change la taille des registres

MX: Word de la forme $0000, $0001, $0100 ou $0101.
Octet de poids fort: M
: : : faible: X (0=16b, 1=8b)

- AddReloc(Seg,Off,Size,Disp,Shift,rSeg,rOff): Relocation.

Seg/Off: Endroit où placer la relocation
Size: Nombre d'octets de la reference
Disp: Deplacement [-80,+7F]
Shift: Decalage en bits, <0 vers la droite
rSeg/rOff: Référence

- AddExtReloc(Seg,Off,Size,Disp,Shift,Addr/4): Relocation externe.

Seg/Off,Size,Disp,Shift: Idem que AddReloc
Addr: Adresse de la référence

- AddDataBank(Seg,Off,0,DBR): Change la valeur du DBR.

Seg/Off:	Endroit où effectuer le changement.
0:	Réservé - Mis a zéro
DBR:	Nouvelle valeur du DBR. Pour B=K, DBR=$8000.

- AddOffset(Seg,Off,0,0): Place un offset.

Seg/Off:	Endroit où placer l'offset
0:	Réservé - Mis a zéro Devrait servir a traiter differents types d'offsets, voire les differences entre labels (Label2-Label1 par ex).

- AddComment(Seg,Off,@String): Place un commentaire

Seg/Off:	Endroit où placer le commentaire.
@String:	Pointeur sur la pString du commentaire.

==========

Il est normalement livré avec TFBD des samples d'extensions avec leurs sources Merlin; n'hésitez pas a aller y faire un petit tour pour voir comment ça marche, c'est fait pour ça.

==========

That's all folks...

==========