Benjamin L.G Torno

Manuel d'utilisation du logiciel

COMPACTILISP

version 1.1 PRO

CAPCPO

Publié le 21 mars 2014.

Note : tout au long du Manuel, il est fait usage du terme « compilation », qui ne renvoie pas à la signification classique qui lui est attribuée en informatique: la conversion d'un ensemble de codes sources en langage machine. Le terme compilation est utilisé pour son sens courant qui est le regroupement d'un ensemble d'ouvrages (fichiers codes sources) en un seul ouvrage (l'ensemble des codes sources regroupés). Il signifie autant l'action de « compiler », que le résultat de cette dernière, une « compilation ».

Important: diverses protections empêchent le logiciel d'écraser un fichier source mais celles-ci ne sont pas infaillibles. Aussi il est vivement conseillé de placer chaque fichier projet de compilation dans un répertoire spécifique.

Sommaire

1. Description du logiciel

CompactiLISP est un logiciel qui permet de rendre difficilement déchiffrable un ensemble de codes sources AutoLISP et VLISP, de les compacter, et de les réunir en un seul fichier LISP plus facile à encoder et à charger dans la machine virtuelle AutoLISP.

CompactiLISP 1.1 est disponible à ce jour pour Bricscad et Zwcad, en français et en anglais. Il se décline en deux versions: une version simple et une version professionnelle. Quelle que soit la version, CompactiLISP offre un rapport complet sur l'utilisation des symboles employés dans l'ensemble des codes sources. Cela permettant de développer des logiciels plus sûrs et optimisés.

Le cryptage des symboles se fait de manière sécurisée afin de garantir le bon fonctionnement des programmes.

Une fois la compilation des codes sources enregistrée dans un unique fichier, il est alors facile de crypter ce dernier au format DES de Bricscad, ou ZEL de Zwcad. (On peut aussi utiliser le compilateur VLISP d'Autocad pour crypter au format FAS).

1.1 Listes des fonctionnalités communes aux deux versions

- Regroupement d'un ensemble de fichiers LISP en un seul fichier LISP.
- Suppression des commentaires.
- Suppression des indentations, des retours à la ligne et des espaces superflus.
- Identification des symboles AutoLISP, VLISP, et des symboles utilisés par le logiciel de CAO.
- Cryptage des symboles mis à part:
  - ceux d'AutoLISP et VLISP;
  - ceux destinés à être utilisés en tant que nom de commande;
  - ceux déclarés comme étant protégés;
  - ceux qui employés comme nom de fonction ne seraient pas définis par un DEFUN;
  - ceux qui sont quotés ou utilisés à l'intérieur d'expressions quotées.
- Deux méthodes de cryptage des symboles: numéroté ou aléatoire de 4 lettres de A à Z.
- Préfixage des symboles cryptés par une chaîne de caractères spécifiée par l'utilisateur ou par défaut par le nom du projet de compilation.
- Codage possible des chaînes de caractères en octal.
- Compression et intégration des codes sources DCL dans l'exécutable.
- Intégration de fichiers texte dans l'exécutable.
- Affichage d'un rapport de compilation dans la ligne de commande, et enregistré dans un fichier log:

  - compte rendu pour chaque expression AutoLISP, des variables et fonctions, globales et locales utilisées.
  - avertissement lorsqu'une variable globale est utilisée aussi comme variable locale.
  - avertissement lorsqu'une fonction est indéfinie.
  - avertissement lorsqu'une fonction est définie plus d'une fois.
  - avertissement lorsqu'une variable locale n'est pas utilisée.
  - avertissement lorsqu'un symbole AutoLISP (ou VLISP) est redéfini.
  - message de confirmation lorsqu'un symbole est préservé, ou protégé.

- Vérification de la syntaxe des codes sources:
  - vérification du bon nombre de parenthèses.
  - détection des chaînes de caractères ouvertes (dans la mesure du possible).
  - vérification qu'une expression SETQ a un nombre pair d'arguments et que le premier argument de chaque paire est bien un symbole.
  - vérification que les expressions SET, APPLY ou EQ ont bien deux arguments.
  - vérification que le premier argument d'un FOREACH est un symbole.
  - vérification qu'une expression = à bien un nombre d'argument supérieur ou égale à 1
  - vérification qu'une expression EQUAL à un nombre d'argument égale à 2 ou à 3.

- Optimisation des codes sources:
  - message d'alerte si définition de symboles locaux inutiles
  - retire les PROGN inutiles:
    - dans les boucles FOREACH, WHILE, et REPEAT;
    - dans les arguments d'une expression COND;
    - dans le corps d'une expression DEFUN;
    - dans le corps d'une expression LAMBDA;
    - qui n'ont qu'une seule expression ou qui n'ont aucune expression (remplacés par un NIL);
  - comparaison du type =, EQ, ou EQUAL d'un symbole, ou une expression, avec NIL, remplacée par une expression (NULL ...).

1.2 Listes des fonctionnalités de la version PRO

- Liste de l'ensemble des fonctions définies et liste des fonctions principales, ce qui permet de repérer entre autres les définitions de fonctions inutiles.
- Compilation conditionnelle.
- Protection optionnelle contre la trace des fonctions.
- Intégration optionnelle d'un module de gestion de l'installation du programme.
- Intégration optionnelle d'un module de gestion de codes d'activation et version d'essai.
- Cryptage optionnel de la compilation dans un format compatible Autocad, Zwcad, Bricscad et Intellicad.

1.3 Versions d'essai de CompactiLISP

Les versions d'essai sont limitées aux niveaux des fonctionnalités disponibles, la taille des codes sources à compiler, en temps d'utilisation et en nombre d'utilisations. Voici les listes des limitations:

- Version simple:
  - Les symboles sont préfixés uniquement avec le nom du fichier compilation.
  - Le cryptage des symboles se fait uniquement par la méthode de numérotation.
  - La taille globale des codes sources est limitée à 1000 évaluations (soit environ 10ko).

- Version PRO:
  - Les symboles sont préfixés uniquement avec le nom du fichier compilation
  - Le cryptage des symboles se fait uniquement par la méthode de numérotation.
  - La taille globale des codes sources est limitée à 3200 évaluations (soit environ 32ko).
  - Absence de protection contre la trace.
  - Absence de module d'activation.
  - Absence de cryptage de la compilation.

2. Installation

Pour installer CompactiLISP, on peut:

- soit charger l'exécutable LISP qui convient parmi ceux proposés dans le répertoire "LISP/" de la distribution à l'aide de la commande APPLOAD du logiciel de CAO.

- soit utiliser l'expression LISP suivante qui permet d'amorcer l'installation en indiquant directement l'emplacement du fichier exécutable. L'expression s'exécute en la copiant collant dans la ligne de commande du logiciel de CAO et en validant par la touche ENTRÉE:

```
(if
(setq capcpo-f (getfiled "Charger le fichier LISP" "compactiLISP-" "" 2))
      (load capcpo-f)
      (alert "erreur ou abandon!")
)
```

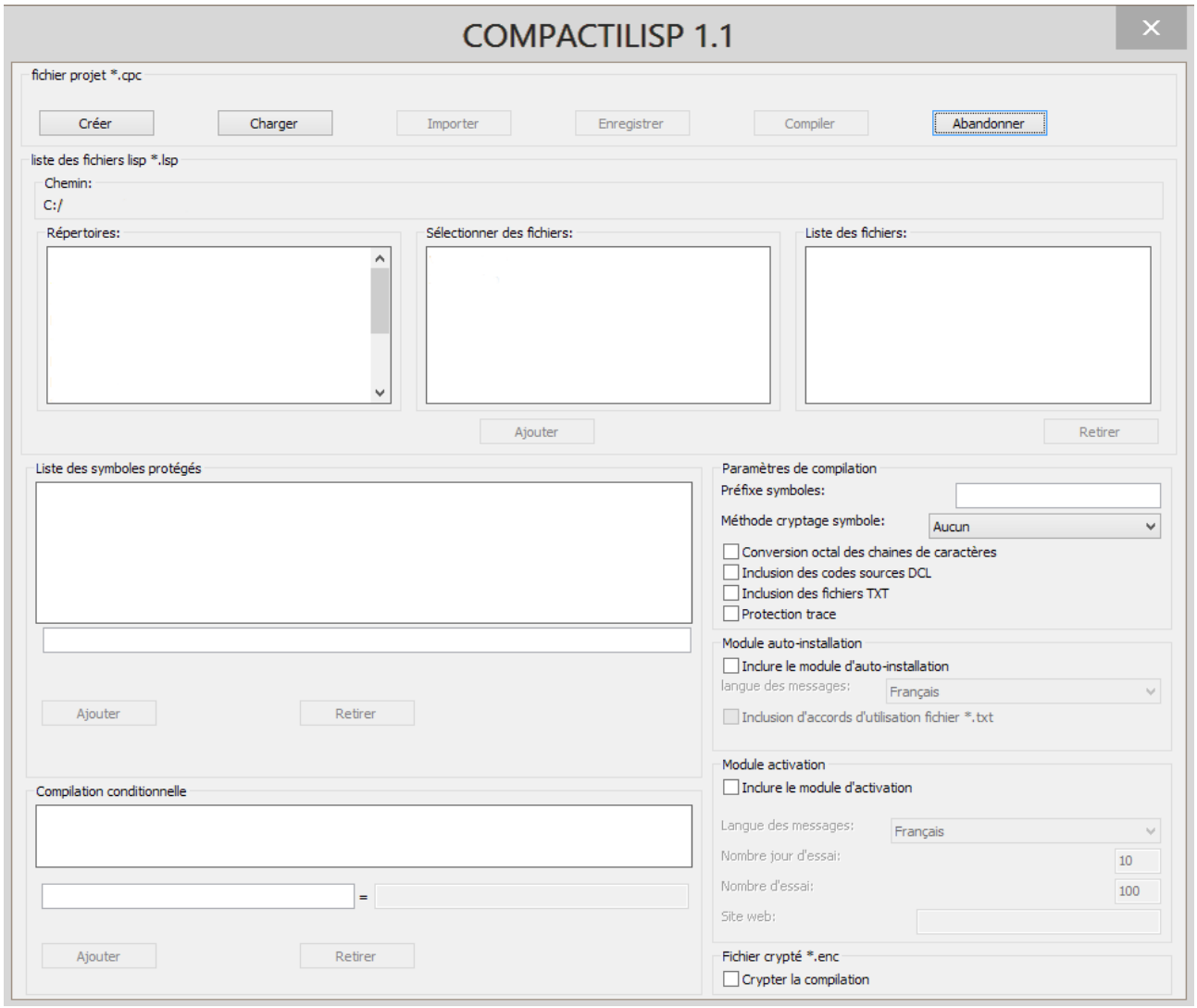
Il suffit alors d'aller chercher le fichier exécutable du logiciel qui convient comme avec la commande APPLOAD.

Pour déterminer l'exécutable qui convient, le nom de ce dernier contient, outre les deux numéros de version, la langue des messages du logiciel, et le logiciel de CAO pour lequel il est destiné, suivant le format compactiLISP-1-1-xx-yyyyyy. Où xx est égale soit à "fr" pour version française soit à "en" pour version anglaise, yyyyyy étant le nom du logiciel de CAO.

3. Utilisation du logiciel

3.1 Utilisation de la boîte de dialogue

Une première façon simple de se servir du logiciel consiste à utiliser sa boîte de dialogue en lançant la commande COMPACTILISP dans la ligne de commande. Effectivement la boîte de dialogue permet de saisir facilement l'ensemble des options de compilation d'un fichier projet. (voir ci-dessous la capture d'écran de la boite de dialogue).



Sans rentrer dans le détail des divers paramètres de compilation, voici succinctement l'ensemble des fonctionnalités de la boîte de dialogue.

En haut se trouve une première rangée de boutons qui permettent dans l'ordre:

- de créer un nouveau projet compactiLISP, fichier à l'extension ".cpc" (ajoutée automatiquement).
- de charger un ancien projet compactiLISP.
- d'importer l'ensemble des fichiers LISP d'un projet de compilation VLISP Autocad, fichier à l'extension ".prj", ainsi que les éventuels symboles protégés spécifiés dans un fichier à l'extension ".gld", du même nom que le fichier projet, et situé dans le même répertoire.
- de sauvegarder l'ensemble des paramètres de compilation dans un fichier projet compactiLISP (soit créé soit chargé préalablement).
- de sauvegarder et lancer la compilation avec les paramètres de compilation en cours dans la boîte de dialogue.

En dessous de la première rangée se trouve une rangée de trois boîtes permettant de lister la liste des répertoires inclus dans le répertoire en cours, la liste des fichiers inclus dans le répertoire en cours, la liste des noms de fichiers retenus pour la compilation. Deux boutons, l'un Ajouter et l'autre Retirer, permettent l'ajout et le retrait simultané d'un ou plusieurs fichiers à compiler.

Viens ensuite la boîte pour spécifier les symboles protégés. On peut ajouter une liste de plusieurs symboles à la fois, tant que ces derniers sont séparés par des espaces, des tabulations, ou des retours à la ligne, en copiant collant le texte de la liste dans le champ d'édition prévu à cet effet. Une fois la liste saisie, il faut valider cette dernière en tapant la touche ENTRÉE. Le bouton d'ajout devient alors actif, et l'on peut ajouter les symboles en appuyant dessus. (Noter que les doublons sont systématiquement supprimés.) Pour retirer des symboles de la liste il suffit d'en sélectionner un ou plusieurs et d'appuyer sur le bouton Retirer.

Pour ceux qui ont la version PRO, une boîte similaire à celle des symboles protégés et en dessous de cette dernière permet de spécifier les conditions de compilation conditionnelle. Chaque condition de compilation nécessite un symbole et une valeur textuelle pour le symbole en question. Il y a donc deux champs d'édition. Celui à gauche, le premier permet de spécifier un symbole de compilation conditionnelle. Le deuxième champ d'édition, celui de droite, permet de saisir une valeur textuelle. Il ne devient actif qu'une fois le symbole de compilation conditionnelle saisi et validé par un appui sur la touche ENTRÉE. De même le bouton Ajouter ne devient actif qu'après une saisie valide d'une valeur textuelle dans le deuxième champ. Comme pour les symboles protégés, les conditions de compilation conditionnelle qui seraient redondantes sont automatiquement supprimées. Pour retirer une condition de compilation conditionnelle, il suffit d'en sélectionner une ou plusieurs dans la liste et d'appuyer sur le bouton Retirer.

Vient enfin, l'ensemble des options de compilation alignées dans une colonne sur le côté droit de la boîte de dialogue. La saisie de ces dernières ne présente pas de difficultés particulières aussi, le lecteur est invité à se reporter à la section suivante pour connaître leur

utilité.

3.2 Utilisation de la fonction en ligne de commande

L'exécution du logiciel peut également être lancée à partir d'une fonction en ligne de commande. Cette fonction porte le même nom que la commande du logiciel, COMPACTILISP. La fonction par contre s'appelle avec des parenthèses ouverte et fermée puisque c'est une fonction AutoLISP. Elle prend deux arguments : le chemin qui mène au fichier projet et le nom du fichier projet. Voici son prototype à saisir en ligne de commande:

```
(compactiLISP chemin-fichier-cpc nom-fichier-cpc)
```

La fonction en ligne de commande peut être pratique si l'on veut compiler plusieurs projets les uns à la suite des autres. Il faut bien sûr au préalable spécifier les paramètres de compilation dans un fichier projet, qui doit se terminer par l'extension « .cpc ». On peut, pour se faire, utiliser la boîte de dialogue du logiciel pour créer un fichier projet, comme on peut également créer directement un fichier projet à l'aide d'un éditeur de texte tel que Blocnote ou Notepad++, à condition de respecter un prototype, disponible dans le répertoire « Ressource » de l'installation du logiciel. Ce dernier contient toutes les informations nécessaires pour comprendre comment créer des fichiers projets manuellement. Un fichier projet étant en fait tout simplement un fichier AutoLISP décrivant une liste d'associations, aussi personne ne devrait éprouver de difficultés pour comprendre sa syntaxe, il suffit juste d'utiliser les bons mots-clés au début des listes et de renseigner les bons types de données associées. Or, il n'y a que trois types possibles : des chaînes de caractères (entre guillemets), des entiers ou bien tout simplement le symbole NIL.

4. Description des divers paramètres de compilation

4.1 Symboles protégés

Si un symbole n'est pas un nom de commande précédé d'un « C : »; n'est pas utilisé à l'intérieur d'une expression quotée; et n'est pas une fonction indéfinie; alors il sera crypté à la compilation.

Les symboles locaux sont cryptés séparément des symboles globaux. De plus, comme il est impossible de prévoir par quel symbole, un symbole sera remplacé; et que le bon fonctionnement d'un programme peut nécessiter qu'un symbole ne soit pas crypté; il peut s'avérer nécessaire de spécifier qu'un symbole soit protégé, c'est-à-dire laissé intact dans la compilation. Tant les symboles globaux, que locaux peuvent être protégés. (d'ailleurs au passage, un symbole peut être à la fois globale et locale. Pratique de programmation qu'il vaut mieux éviter pour se prémunir d'avoir des erreurs difficilement repérable à l'exécution.)

Un exemple typique est celui des expressions LISP utilisées pour les actions des objets DCL, tel que les boutons, les champs d'édition, etc. La plupart du temps ces expressions utilisent des symboles globaux qui permettent de communiquer avec le programme principal. Or, les

expressions LISP relatives aux actions DCL sont enregistrées sous la forme de chaîne de caractères à lire (fonction READ) et à évaluer (fonction EVAL) à l'exécution. Ces expressions contenues dans des chaînes de caractères n'étant pas analysées par CompactiLISP version 1.1, il faut protéger tous les symboles globaux qui y seraient employés à des fins de communication. Sans cela, ces mêmes symboles globaux se retrouvent cryptés dans le corps du programme mais pas à l'intérieur des chaînes de caractères des actions DCL. La communication entre le module DCL d'un programme et le programme lui-même est alors rompue.

Un autre exemple est le cas de variables globales d'un programme qui sont données en accès à l'utilisateur par le biais d'un fichier d'initialisation. Si l'on ne protège pas les symboles de ces variables, le fichier d'initialisation ne sert plus à rien !

Concernant les expressions quotées, celles-ci sont analysées par le logiciel, mais étant donné les possibilités très souples des langages LISP, il a été jugé préférable de protéger systématiquement les symboles qu'elles contiennent afin de ne pas avoir de surprise et d'être sûr que le programme fonctionnera correctement après la compilation. Pour s'en convaincre il suffit de prendre l'exemple suivant (qui peut se décliner sous une infinité de formes):

```
(= (type SYM) 'SYM)
```

Si par exemple on crypte alors le symbole SYM par μ0 et que l'on remplace également dans l'expression quotée, l'expression devient :

```
(= (type μ0) 'μ0)
```

qui bien sûr ne donne pas le même résultat dès lors que la donnée affectée au symbole μ0 est elle-même un symbole! C'était un exemple simple, mais prenons une expression LAMBDA quotée faisant appel à un symbole localisé, qui soit également un symbole globale, il est impossible de savoir à l'avance au quel des deux symboles (identique) on aurra à faire (même sachant que la localisation prévaut sur la globalité). Tout dépend en fait, à quelle niveau l'expression LAMBDA est exécutée.

Aussi la stratégie qui a été retenue pour cette première version du logiciel, est la sécurisation des cryptages pour garantir le bon fonctionnement des programmes compilés. Des améliorations seront apportées dans les versions futures si elles sont pertinentes et n'entravent pas le bon fonctionnement des programmes, et a priori avec un choix à faire par l'utilisateur.

Donc dans tous les cas où un symbole a besoin de rester identique à lui-même, il faut le spécifier au logiciel par l'intermédiaire de la liste des symboles protégés. Si l'un de ces symboles est par ailleurs utilisé dans une expression quotée, on peut alors éventuellement omettre de le déclarer comme symbole protégé puisqu'il sera automatiquement préservé.

4.2 Préfixe des symboles cryptés

Afin de cloisonner les espaces de nom de symboles entre les différents programmes LISP, il est recommandé d'utiliser un préfixe pour les symboles globaux (le préfixe étant inutile pour les symboles localisés). Le préfixe n'a pas besoin d'être très grand, quatre ou cinq lettres

suffisent amplement, comme « #°_°# ». Choisissez seulement des lettres autorisées pour les symboles AutoLISP, donc ne pas utiliser les lettres suivantes : « ! », « (« , «) », « " », « ; », « ' ». Le « . » étant à éviter également en première position avec Bricscad ou partout avec Zwcad. Les chiffres pouvant être utilisés à condition qu'il y est au moins une lettre valide.

Si vous ne spécifiez pas de préfixe alors le nom du fichier projet, qui est le nom implicite de projet, est alors utilisé comme préfixe accolé de la lettre « µ ». Dans ce dernier cas votre préfixe risque d'être un peu trop long ce qui ralentira l'exécution du programme compilé. (Remarque : avec zwcad le nom du fichier cpc ne doit pas comporter de "." autre que celui précédant l'extension, à moins bien sûr, de définir un préfixe pour les symboles ne comportant pas de "." également.)

A noter, par ailleurs que pour brouiller plus efficacement le code compilé, les symboles localisés sont également préfixés, cela n'entraînant pas de réelles modifications de l'efficacité du programme compilé mais rend beaucoup plus difficile la lecture du code source compilé.

4.3 Méthode de cryptage des symboles

La version simple propose les deux méthodes de cryptage suivantes :

- La numérotation des symboles : un symbole est remplacé par un numéro unique fourni par un compteur globale. Les symboles cryptés étant tous préfixés, on obtient des symboles AutoLISP valides.
- Le cryptage aléatoire A-Z : au préfixe est ajouté une séquence de 4 lettres aléatoires parmi les lettres majuscules de A à Z. Ce qui donne une combinatoire de 456976 symboles possibles. Chaque séquence aléatoire générée est comparée à celles précédemment générées afin d'éviter le cas rarissime, mais possible, d'un doublon. Les séquences aléatoires générées sont ainsi uniques.

La version PRO propose une troisième méthode de cryptage des symboles :

- le cryptage avec des séquences de 8 caractères invisibles . La lecture par un être humain du code compilé est ainsi impossible puisque l'on obtient suivant le logiciel de traitement texte utilisé du code qui peut ressembler à :

  ```
  (#°_°# #°_°# (#°_°# (#°_°# #°_°# #°_°# #°_°#)))
  ```

  ou encore :

  ```
  (#°_°#??? #°_°#??? (#°_°#??? (#°_°#??? #°_°#??? #°_°#??? #°_°#???)))
  ```

  La combinatoire des séquences cryptées invisibles est de 390625, et à nouveau les séquences générées sont uniques.

A noter enfin que l'on peut choisir de ne crypter aucun symbole en choisissant la méthode AUCUN dans la liste déroulante de la boîte de dialogue, ou en spécifiant "AUCUN" comme option de cryptage des symboles dans le fichier projet.

4.4 Conversion des chaînes de caractères en octal

L'interpréteur AutoLISP peut lire des chaînes de caractères passées sous forme octale. Par exemple : la chaîne "A" est équivalente à la chaîne octale "\101". Avec l'option de conversion des chaînes de caractères, ces dernières sont remplacées par leur équivalent en octal, permettant de rendre plus confuse la lecture des codes sources ainsi compilés. (A noter que cela entraîne un accroissement de la taille des chaînes de caractères et donc un accroissement de la taille globale de la compilation.)

4.5 Inclusion des codes sources DCL

Les codes sources DCL peuvent être inclus dans la compilation, évitant ainsi des paramétrages fastidieux lors de l'installation du programme. L'utilisation de cette option est très simple, elle se fonde sur la fonction AutoLISP LOAD_DIALOG qui prend en argument un nom de fichier DCL. Pour que l'inclusion du fichier DCL soit possible il suffit juste de spécifier en brut le nom de fichier DCL à la fonction LOAD_DIALOG. Par exemple :

```
(LOAD_DIALOG "C:/.../fichier-dcl.dcl")
```

Ainsi le logiciel peut aller chercher le fichier DCL et l'intégrer dans la compilation. À l'exécution de cette dernière tout se passe comme si le fichier avait été chargé depuis son emplacement d'origine.

Le fichier DCL est de plus compacté pour gagner de la place et du temps lors du chargement.

4.6 Inclusion des fichiers textes

L'inclusion de fichiers texte ascii est également possible. Cela fonctionne comme pour les fichiers DCL, en utilisant une fonction générique LOAD_TEXT dont le code source est disponible dans le fichier load_text.lsp situé dans le répertoire Ressource de l'installation du logiciel. Il suffit alors d'écrire quelque part dans ses codes sources un appel à la fonction LOAD_TEXT comme ceci :

```
(LOAD_TEXT "C:/.../fichier-text.txt")
```

La fonction LOAD_TEXT renvoyant un liste contenant les chaines de caractères présentes dans un fichier texte, une chaîne par ligne de texte.

Avant la compilation, il suffit de charger la fonction LOAD_TEXT avec les codes sources d'un programme pour pouvoir exécuter ces derniers. À la compilation, les chaînes de caractères d'un fichier texte sont incluses dans les codes sources, qui n'ont donc plus besoin d'être chargées à l'exécution. Il n'est inutile, par conséquent, de joindre les codes sources de la fonction LOAD_TEXT à la compilation.

4.7 Compilation conditionnelle

Lorsque l'on développe un logiciel on a souvent besoin de prendre en compte des paramètres comme la langue des messages, le logiciel de CAO utilisé, un type de version etc. On peut alors développer des versions concurrentes d'un logiciel, mais cela est fastidieux à gérer. On peut aussi choisir d'intégrer des structures de contrôle, expressions IF et COND, pour gérer

divers paramètres d'exécution. Cette deuxième possibilité, qui présente l'inconvénient d'alourdir un logiciel, reste cependant la meilleure solution pour développer sereinement.

La compilation conditionnelle permet alors de simplifier toutes les structures de contrôle dédiées aux paramètres d'exécution du logiciel, pour ne garder que les codes sources dont on a besoin dans la compilation.

Par exemple le code suivant permet d'afficher un texte selon un paramètre de langue :

```
(COND
((= LANGUE "FR")
      (PRINC "\nBonjour!")
)
((= LANGUE "EN")
      (PRINC "\nHello!")
)
)
```

Si l'on spécifie une condition de compilation sur le symbole LANGUE en fixant par exemple la valeur du symbole à la chaîne de caractères `"FR"`, alors dans la compilation ne sera retenu que le code :

```
(PRINC "\nBonjour!")
```

Les conditions de compilation sont très pratiques, car elles permettent de développer un programme et de le tester comme à l'habitude, puis de simplifier les codes sources lors de la phase de compilation.

Règles pour l'utilisation de la compilation conditionnelle.

Voici les quelques règles à connaître pour bien utiliser les conditions de compilation.

En premier lieu, il faut avoir à l'esprit que le logiciel identifie les expressions de test de la forme `(= SYMBOLE "VALEUR")` dans les structures de contrôle IF et COND. Si dans l'expression de test se trouve un symbole de compilation conditionnelle, préalablement spécifié dans le fichier projet, le test de la structure de contrôle est alors pour ainsi dire pré évalué. Du résultat de la pré évaluation du test, dépend l'intégration, ou pas, du bloc de code qui lui est associé.

Voici tous les cas de figure possible :

```
....version d'évaluation: partie restreinte...
```

4.8 Protection contre la trace

Voici une option supplémentaire pour à la fois rendre encore plus difficile la lecture de la compilation, et protéger le programme compilé contre la trace. En effet, il est très facile dans le logiciel de CAO de demander que l'ensemble des fonctions définies, AutoLISP et fonctions utilisateurs, soit tracé par l'intermédiaire de la fonction TRACE. Il est alors possible de remonter la logique d'un programme en suivant le fil des appels de fonctions.

```
CompactiLISP 1.1
```

Problème que ne connaissent pas ceux qui développent des logiciels avec Autocad en les compilant au format FAS, mais avec Zwcad et Bricscad, le format FAS n'est pas disponible. Bien que Bricscad empêche la trace de certaines fonctions AutoLISP et VLISP, une protection contre la trace est toujours un plus pour protéger ses codes sources, notamment avec Zwcad.

Le cryptage supplémentaire des symboles de fonctions rend par ailleurs, la lecture des codes encore plus difficile (mais non pas indéchiffrable pour qui aurait accès au code source de la compilation).

Comment fonctionne la protection contre la trace ?

```
....version d'évaluation: partie restreinte...
```

4.9 Module d'installation

L'option du module d'installation est vraiment très utile et fait gagner beaucoup de temps, elle permet de rendre un programme auto-installable. En effet, lors du chargement d'un programme compilé avec le module d'installation, l'installation de ce dernier se lance automatiquement.

En quoi consiste l'installation? Si vous avez chargé CompactiLISP, l'installation de ce dernier s'est automatiquement lancée et vous avez donc l'illustration de ce que peut accomplir le module d'installation, dont voici les fonctionnalités :

1. Demande de confirmation pour l'installation d'un programme.
2. Demande de confirmation concernant la langue d'un programme.
3. Affichage éventuel du texte des accords d'utilisation du programme avec demande d'acceptation de ces derniers.
4. Paramétrage des fichiers d'initialisation du logiciel de CAO pour que le logiciel puisse être chargé soit manuellement, soit systématiquement, soit sur demande (fonction AUTOLOAD d'AutoLISP).

Règles pour l'utilisation du module d'installation:

Pour utiliser le module d'installation il suffit juste de respecter les règles suivantes :

- Les fichiers LISP exécutables doivent être placés dans un répertoire « LISP/ » situé dans le répertoire principal de l'installation du logiciel.
- Le nom de projet, et donc nom du fichier compilé, doit être constitué ainsi en lettres minuscules :
  - un nom de programme, qui est aussi le nom de la commande principale du programme ;
  - un tiret « - »
  - un numéro majeur de version
  - un tiret « - »
  - un numéro mineur de version
  - un tiret « - »

- une chaîne de caractères identifiant la langue du logiciel : « fr » pour français, « en » pour anglais, « int » si pas de langue spécifique
- un tiret « - »
- une chaîne de caractères identifiant le logiciel de CAO : « zwcad », « bricscad » ou « cad » si pas de logiciel de CAO spécifique.
- l'extension « .lsp », ou « .des » pour Bricscad et « .zel » pour Zwcad.

Ce qui donne un nom de projet la forme : nom_programme-x-x-langue-xxcad.xxx, le nom de programme ne devant pas comporter de tiret « - ». (il faut aussi retirer le « .cpc » ajouté automatiquement dans le nom du fichier compilation pour des raisons de préservation des codes sources initiaux dans le cas où le nom de projet correspondrait au nom de l'un de ces derniers)

- Intégrer dans la distribution du programme un fichier à-lire.txt comme celui présent dans la distribution de compactiLISP
- Concernant la langue des messages du module d'installation, il suffit de spécifier celle-ci dans le fichier projet au moment de la compilation
- Ainsi que concernant les accords d'utilisation à afficher au moment de l'installation, il suffit de spécifier l'emplacement du fichier texte contenant ces derniers, ils seront inclus dans la compilation.

Une fois qu'un programme est installé, une variable initialisée à chaque ouverture d'un dessin dans le logiciel de CAO, permet de ne plus procéder à l'installation de ce programme.

De même une variable globale est affectée du chemin qui mène à l'installation du logiciel sur le disque de l'ordinateur. Cette variable globale est le symbole suivant NOM_DU_PROGRAMME-CHEMIN-INSTALLATION, ou il suffit d'expliciter le nom du programme parce qu'il vaut pour obtenir le chemin de l'installation. Cette variable est donc très pratique pour permettre aux programmes d'accéder aux différentes ressources disponibles dans leur répertoire d'installation. Ce peut être des programmes secondaires à charger comme des modules vba, des fichiers dcl, des documents à afficher, des fichiers tableurs, des bases de données, etc.

Désinstallation

Un programme de désinstallation spécifique au programme compilé est également automatiquement généré et stocké dans un fichier desinstaller-nom_du_programme.lsp. Il est souhaitable de crypter aussi ce fichier au format DES ou ZEL au même titre que le fichier de la compilation, afin d'en protéger les codes sources.

4.10 Module d'activation

L'option pour le module d'activation est également vraiment très pratique et fait gagner beaucoup de temps. Ce module gère version d'essai et codes d'activation pour le programme dans lequel il est intégré. Il nécessite le renseignement de quatre paramètres:

- la langue des messages du module d'activation ;
- un site internet pour identifier le développeur du programme et permettre aux

utilisateurs de le contacter pour acheter un code d'activation ;
- un nombre de jours pour la durée d'utilisation de la version d'essai.
- un nombre d'utilisations limite pour la version d'essai.

Ainsi passé le délai en nombre de jours ou en nombre d'utilisations, la version d'essai ne sera plus active et l'utilisateur est contraint d'acheter un code d'activation s'il souhaite continuer à utiliser le logiciel.

```
....version d'évaluation: partie restreinte...
```

Un générateur de code d'activation pour ses propres programmes est fourni lors de l'achat de la version PRO.

4.11 Cryptage de la compilation

Voici pour finir une dernière option disponible avec le logiciel CompactiLISP PRO, l'option de cryptage de la compilation. Cette option permet de crypter la compilation dans un format compatible avec Zwcad, Bricscad, mais aussi a priori tous les Intellicad et Autocad. Pour peu que le programme compilé soit compatible avec tous ces logiciels de CAO, il pourra alors être chargé sous une forme cryptée sur l'ensemble de ces logiciels. Ce qui peut être bien pratique au bout du compte pour les personnes qui change souvent de station de travail.

Cependant quelques bémols doivent être mentionnés ici:

- Tout d'abord le format de cryptage proposé est celui qui existait avec les premières versions d'Autocad. `....version d'évaluation: partie restreinte...`
- En second lieu, l'algorithme de cryptage est fastidieux, et implémenté en AutoLISP il s'avère long en exécution dès que la taille de la compilation dépasse 32 ko. Il est d'ailleurs possible que l'algorithme de cryptage tel qu'il a été implémenté n'aboutisse pas si le fichier compilé est trop volumineux.
- Le fichier crypté nécessite de pouvoir écrire des nombres binaires sur 8 bits, ce qui n'est pas possible avec la fonction standard d'AutoLISP WRITE-CHAR pour l'écriture dans un fichier d'un entier sur 8 bits (un caractère ASCII). Alors, pour pallier à cela, CompactiLISP fait appel à une fonction VBA avec Zwcad, et fait appel à la fonction VLE-WRITE-UINT8 sous Bricscad, mais seulement à partir de la version 14.

5. Limitations

Voici la liste des limitations de CompactiLISP version 1.1. Le logiciel:

- ne détecte pas la définition d'une fonction qui serait définie par l'intermédiaire de la fonction LAMBDA, tel que (setq f (lambda ...)), cela entraîne seulement l'apparition

d'un éventuel avertissement de non définition, si le symbole f était utilisé comme fonction dans une expression LISP.

- détecte qu'une fonction est définie à plusieurs reprises seulement lorsqu'il s'agit de définitions de fonctions indépendantes les unes des autres, pas lorsque celles-ci sont imbriquées ensemble comme (defun f () (defun f () nil)).

- ne prend pas en compte les chaînes de caractères qui seraient passés sous forme octale "\nnn" dans les codes sources.

- ne prend pas en compte la localisation des variables dans les expressions LAMBDA ou DEFUN qui seraient quotées, puisque les expressions quotées sont analysées seulement en tant que liste d'atomes.

- , contrairement au compilateur VLISP qui conserve les symboles globaux (autres que les fonctions internalisables...), crypte les noms des variables globales à moins de les protéger.

Certains logiciels de CAO comme Intellicad renvoient les codes sources autolisp dans la ligne de commande, il est préférable de ne pas distribuer de logiciels commerciaux pour ces logiciels de CAO, même sous format encrypté.