Manuel des Extensions SuperBOL à GnuCOBOL

Version 1.0: Le Guide Ultime

L'équipe SuperBOL de Titagone-OCamlPro

26 février 2026

Table des matières

Ce manuel est disponible :

- En tant que site web HTML: en Français ou en Anglais
- En tant que fichier PDF : en Français ou en Anglais

1 Avant-propos

L'équipe SuperBOL contribue des extensions au compilateur libre et open-source GnuCOBOL, pour ses clients.

Certaines de ces extensions ne sont pas publiées dans la version officielle du compilateur. Elles ne sont disponibles que pour les clients adhérant à la Souscription SuperBOL, qui donne accès à tous les dépôts open-source gérés par l'équipe SuperBOL.

Ce document décrit ces extensions.

2 Introduction

Bienvenue dans ce manuel d’utilisation.

3 Les Extensions SuperBOL

3.1 Dépôts de code

L'équipe SuperBOL maintient normalement plusieurs dépôts de code pour chaque client:

- Le dépôt **gnucobol-superbol** est un dépôt GnuCOBOL commun à tous les clients de la Souscription SuperBOL. Ce dépôt contient normalement l'ensemble des contributions apportées pour les différents clients.

  Les branches suivantes sont disponibles dans notre dépôt:

  - **gnucobol-superbol-3.x**: nos contributions à GnuCOBOL 3
  - `gnucobol-superbol-4.x`: nos contributions à GnuCOBOL 4 (non utilisée actuellement)
  - `gitside-gnucobol-3.x`: la branche officielle de GnuCOBOL 3 que nous suivons
  - `gitside-master`: la branche officielle de GnuCOBOL 4 que nous suivons
- Le dépôt **gnucobol-$CLIENT** est un dépôt GnuCOBOL spécifique à un client. Il contient la version de GnuCOBOL SuperBOL maintenue pour ce client. Il peut contenir des contributions temporaires supplémentaires (dévelopées spécifiquement pour ce client et non encore propagées au dépôt commun), ou ne pas contenir certaines contributions (pour limiter l'introduction d'incompatibilités).
- Le dépôt **gnucobol-config-$CLIENT** est un dépôt contenant parfois une configuration spécifique à l'application d'un client. En effet, certains patrimoines utilisent parfois une combinaison de fonctionalités ne correspondant pas aux configurations par défaut de GnuCOBOL. Nous fournissons alors une configuration spécifique à utiliser pour compiler les sources COBOL du client.

3.2 Extensions disponibles dans votre version

Chaque dépôt pouvant contenir des contributions différentes, il est possible de lister facilement les contributions que contient votre version. Pour cela, il y a 2 possibilités:

- Vous pouvez regarder, dans le dépôt GIT, le contenu du dossier `CHANGES-superbol/` à la racine de GnuCOBOL. Ce dossier contient un fichier par contribution. Vous pouvez alors vérifier si la contribution qui vous intéresse apparaît dans ce dossier.
- Vous pouvez lancer la commande `cobc --superbol-features`, qui affiche la liste des contributions que ce compilateur intègre:

```
$ cobc --superbol-features
cobc (GnuCOBOL) 3.3-dev.0
SuperBOL features:
- 2025-11-20-odo-in-redefines
- 2025-11-25-allow-spaces-between-parens-in-PICTURE-la-MF
... [suite de l'output] ...
```

3.3 Toutes les extensions

- **2025-11-20-odo-in-redefines**: **[MF-COMPAT]** Cette extension autorise l'utilisation de `OCCURS DEPENDING ON` dans un `REDEFINES`, quand la nouvelle option de configuration `odo-in-redefines` est activée.
- **2025-11-25-allow-spaces-between-parens-in-PICTURE-la-MF**: **[MF-COMPAT]** Cette extension utilise autorise la présence **d'espaces entre les parenthèses d'une `PICTURE`, comme `PIC X( 9  **)`, quand la nouvelle option de configuration `space-in-picture`** est activée.
- **2025-11-25-inspect-with-or**: **[MF-COMPAT]** Cette extension permet l'utilisation de `OR` après un `INSPECT ... BEFORE` ou `INSPECT ... AFTER`, comme `INSPECT str TALLYING counter FOR CHARACTERS BEFORE "1" OR "2"`.
- **2025-11-25-new-level-heuristics**: **[MF-COMPAT]** Cette extension permet de tolérer beaucoup plus de points manquants dans la DATA DIVISION, en utilisant une nouvelle heuristique pour la détection des numéros indiquant les niveaux. **Attention:** elle peut introduire des régressions par rapport à des codes fonctionnant déjà avec GnuCOBOL. Ces régressions doivent nous être signalées pour corriger le comportement.
- **2025-11-25-print-token**: Cette extension introduit l'option `-fdump-tokens` qui permet d'aider au debug du parser.
- **2025-11-28-better-out-of-bounds-messages**: **[Ergonomie]** Cette extension améliore les messages d'erreurs en cas de débordement, typiquement pour afficher les valeurs et les bornes.
- **2026-01-12-compatibility-autoconf-2.69**: Cette extension permet d'utiliser des versions d'`autoconf` à partir de 2.69 au lieu de 2.70
- **2026-01-15-fix-sign-normalization-on-MOVE**: **[BUGFIX]** Cette extension corrige un comportement qui empêche une normalisation du signe lors d'un `MOVE` quand un `DISPLAY NUMERIC` n'a pas été signé correctement à l'initialisation.
- **2026-02-05-add-superbol-testsuite**: Cette extension introduit une nouvelle testsuite spécifique à SuperBOL.

- **2026-02-05-shared-superbol-config**: Cette extension introduit une configuration commune pour toutes les options introduites par les extensions de SuperBOL.
- **2026-02-06-superbol-atscript-tests**: Cette extension introduit un nouveau format basé sur `autofonce` pour certains tests de non-regression de SuperBOL.
- **2026-02-07-allow-more-extra-dots**: **[BUGFIX]** Cette extension corrige un comportement erronné de GnuCOBOL qui ne supporte pas des points redondants s'ils sont séparés par des directives de lignes, typiquement introduites par des `COPY`.
- **2026-02-08-allow-unterminated-strings**: **[MF-COMPAT]** Cette extension autorise l'absence de terminateur de chaîne, quand la nouvelle option de configuration `unterminated-string` est activée.
- **2026-02-08-missing-dot-after-SPECIAL-NAMES**: **[MF-COMPAT]** Cette extension autorise l'absence d'un point à la fin des instructions dans `SPECIAL-NAMES`, juste avant `INPUT-OUTPUT`, `DATA DIVISION` ou `PROCEDURE DIVISION`.
- **2026-02-08-missing-dot-before-or-after-FD**: **[MF-COMPAT]** Cette extension autorise l'absence de point dans un champ avant ou après `FD` ou `SD`.
- **2026-02-17-autostop**: **[Ergonomie]**. Cette extension permet d'interrompre une commande COBOL en milieu de batch à l'aide d'une variable d'environnement pour lancer une session de debug dessus. Voir la section Interrupteur de Batch
- **2026-03-10-handle-spzero**: **[MF-COMPAT]** support de compatibilité avec l'option `SPZERO` du compilateur Micro-Focus

4 Interrupteur de batch

Feature: `2026-02-17-autostop`

4.1 Aperçu

L'interrupteur de batch permet de suspendre le lancement d'un programme spécifique dans une chaîne, sans intervention utilisateur. Lorsqu'un programme est ainsi suspendu, un fichier contenant son PID (identifiant de processus) est produit : en utilisant cette information, un débogueur tel que **gdb** ou celui inclus dans **SuperBOL Studio** peut alors être attaché au programme, afin d'analyser son comportement.

4.2 Prérequis

Les programmes à interrompre et analyser doivent être compilés en mode débogage (option -g) avec un compilateur GnuCOBOL modifié. Les fichiers C intermédiaires générés par GnuCOBOL doivent être accessibles au débogueur.

4.3 Fonctionnement

Préalablement au lancement de la chaîne, la liste des programmes à interrompre doit être renseignée dans la variable d'environnement `COB_STOP_JOBS`. Il peut s'agir de noms de programmes tels que spécifiés par une clause `PROGRAM-ID`, ou de noms de fichiers exécutables (pour les programmes compilés comme des exécutables avec l'option `-x` de GnuCOBOL). Les différents noms de programmes doivent être séparés par une virgule.

Le nom du fichier utilisé pour stocker le PID du programme interrompu peut être personnalisé grâce à la variable `COB_JOB_PIDFILE`. En son absence, le fichier sera nommé `$EXECUTABLE-$JOB.pid`, où `$EXECUTABLE` correspond au nom du fichier exécutable (`cobcrun` pour un module), et `$JOB` correspond au nom du programme tel qu'il apparaît dans la clause `PROGRAM-ID`.

La chaîne peut ensuite être exécutée normalement. Lorsqu'elle est interrompue,le débogueur **gdb** peut être attaché au programme à l'aide de la commande :

```
$ gdb -p $(cat fichier.pid)
```

Ou, si l'on préfère passer par **gdbserver** plutôt que gdb :

```
$ gdbserver --attach :port $(cat fichier.pid)
```

Au démarrage de la session de débogage, le programme est arrêté dans un appel à la fonction raise de la librairie standard C. Pour remonter jusque dans le programme COBOL (du moins dans sa traduction en C), on peut utiliser la commande finish (appuyer plusieurs fois sur la touche entrée pour répéter la commande).

Le débogueur inclus dans **SuperBOL Studio** peut également être attaché au programme, en renseignant le PID correspondant (se référer à la documentation de SuperBOL Studio).

4.4 Exemple

On suppose que l'on a un fichier `prog.cob` contenant deux programmes dont les `PROGRAM-ID` sont `prog1` et `prog2`, et un fichier `sousprog.cob` contenant un programme dont le `PROGRAM-ID` est `sousprog`. Le programme `prog1` appelle `prog2` et `sousprog` à l'aide d'un `CALL`.

On compile les programmes comme suit :

```
$ cobc -g -x prog.cob
$ cobc -g -m sousprog.cob
```

On obtient deux fichiers binaires `prog` et `sousprog.so`. L'option `-g` permet d'intégrer les informations de débogage à ces deux fichiers binaires, ainsi que de conserver les fichiers C intermédiaires.

On veut interrompre les appels aux programmes `prog1`, `prog2`, et `sousprog`. Pour ce faire, on configure la variable d'environnement `COB_STOP_JOBS` :

```
$ export COB_STOP_JOBS=prog1,prog2,sousprog
```

A noter que puisque tous les programmes contenus dans le fichier exécutable `prog` sont concernés, on aurait également pu écrire :

```
$ export COB_STOP_JOBS=prog,sousprog
```

On lance alors le programme :

```
$ ./prog
```

Son exécution est alors silencieusement interrompue. A l'aide d'un autre terminal, on cherche le fichier contenant le PID du processus :

```
$ ls *.pid
prog-prog1.pid
```

L'exécution s'est donc bien interrompue sur le programme `prog1`. On attache le débogueur :

```
$ gdb -p $(cat prog-prog1.pid)
```

Le débogueur affiche :

```
Program received signal SIGTSTP, Stopped (user).
__pthread_kill_implementation (no_tid=0, signo=20, threadid=140599531862272)
at ./nptl/pthread_kill.c:44
44    ./nptl/pthread_kill.c: Aucun fichier ou dossier de ce type.
(gdb)
```

A l'aide de la commande finish, on remonte jusqu'au code du programme :

```
(gdb) finish       (appuyer plusieurs fois sur entrée)
```

On obtient :

```
Run till exit from #0  0x00007fdfe1cf1c10 in cob_set_cancel () from /opt/
gnucobol-sb/lib/libcob.so.4
prog1_ (entry=0) at prog.c:185
185      b_2 = 0;
(gdb)
```

A partir de là, on peut explorer l'exécution du programme `prog1`.

Si l'on poursuit l'exécution (commande `continue` de `gdb`), celle-ci va de nouveau s'arrêter automatiquement sur `prog2`, puis `sousprog`, produisant les fichiers `prog-prog2.pid` et `prog-sousprog.pid` (le PID ne change bien évidemment pas, mais la présence du fichier permet de savoir quel programme a été interrompu).

A noter que si `prog.cob` avait été compilé comme un module plutôt que comme un exécutable, les fichiers PID produits se seraient nommés `cobcrun-prog1.pid`, `cobcrun-prog2.pid`, `cobcrun-sousprog.pid`.