Manual of SuperBOL Extensions to GnuCOBOL

Version 1.0: The Ultimate Guide

The SuperBOL Team at Titagone-OCamlPro

February 26th, 2026

Generated on 2026-11-18 from commit superbol-manuals@a5e6f46

Table of contents

1 Forewords 2
2 Introduction 3
3 SuperBOL Extensions 4
  3.1 Code Repositories 4
  3.2 Available Extensions in your version 4
  3.3 All Extensions 5
4 Batch Interrupter 7
  4.1 Overview 7
  4.2 Prerequisites 7
  4.3 Operation 7
  4.4 Example 8

This manual is available:

- As an HTML website: in English or in French
- As a PDF File: in English or in French

Generated on 2026-11-18 from commit superbol-manuals@a5e6f46

1 Forewords

The SuperBOL team contributes extensions to the GnuCOBOL free and open-source compiler for its customers.

Some of these extensions have not been upstreamed to the official version of the compiler. They are only available to customers with the SuperBOL Subscription, that gives access to all internal open-source repositories managed by the SuperBOL team.

These extensions are described in this document.

2 Introduction

Welcome to this user manual.

3 SuperBOL Extensions

3.1 Code Repositories

The SuperBOL team typically maintains several code repositories for each client:

- **`gnucobol-superbol`**: This is a common GnuCOBOL repository for all SuperBOL Subscription clients. This repository normally contains the entirety of the contributions made for various clients.

  The following branches are available in this repository:

  - **gnucobol-superbol-3.x**: our contributions to GnuCOBOL 3
  - **gnucobol-superbol-4.x**: our contributions to GnuCOBOL 4 (currently not used)
  - `gitside-gnucobol-3.x`: the official branch of GnuCOBOL 3 that we are following
  - `gitside-master`: the official branch of GnuCOBOL 4 that we are following

- **`gnucobol-$CLIENT`**: This is a client-specific GnuCOBOL repository. It contains the version of GnuCOBOL SuperBOL maintained for that specific client. It may contain additional temporary contributions (developed specifically for that client and not yet propagated to the common repository) or exclude certain contributions (to limit the introduction of incompatibilities).

- **`gnucobol-config-$CLIENT`**: This repository sometimes contains a specific configuration for a client's application. Indeed, some legacy codebases occasionally use a combination of features that do not match GnuCOBOL's default configurations. In such cases, we provide a specific configuration to be used when compiling the client's COBOL source files.

3.2 Available Extensions in your version

Since each repository can contain different contributions, you can easily list the features included in your version. There are two ways to do this:

- **In the Git repository**: You can check the contents of the `CHANGES-superbol/` folder at the root of GnuCOBOL. This folder contains one file per contribution. You can then verify if the contribution you are interested in appears in this folder.
- **Via the command line**: You can run the command `cobc --superbol-features`, which displays the list of contributions integrated into that compiler:

```
$ 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
... [rest of output] ...
```

3.3 All Extensions

- **2025-11-20-odo-in-redefines**: **[MF-COMPAT]** This extension allows the use of `OCCURS DEPENDING ON` within a `REDEFINES` clause when the new configuration option `odo-in-redefines` is enabled.
- **2025-11-25-allow-spaces-between-parens-in-PICTURE-la-MF**: **[MF-COMPAT]** This extension allows the presence of **spaces between the parentheses of a `PICTURE**`, such as `PIC X( 9 )`, when the new configuration option `space-in-picture` is enabled.
- **2025-11-25-inspect-with-or**: **[MF-COMPAT]** This extension allows the use of `OR` after an `INSPECT ... BEFORE` or `INSPECT ... AFTER`, such as `INSPECT str TALLYING counter FOR CHARACTERS BEFORE "1" OR "2"`.
- **2025-11-25-new-level-heuristics**: **[MF-COMPAT]** This extension allows the compiler to tolerate many more missing periods in the `DATA DIVISION` by using a new heuristic for detecting level numbers. **Warning:** It may introduce regressions compared to code already functioning with GnuCOBOL. These regressions should be reported to us to correct the behavior.
- **2025-11-25-print-token**: This extension introduces the `-fdump-tokens` option to assist in debugging the parser.
- **2025-11-28-better-out-of-bounds-messages**: **[Ergonomics]** This extension improves error messages for overflows/out-of-bounds errors, typically by displaying the actual values and bounds.
- **2026-01-12-compatibility-autoconf-2.69**: This extension allows the use of `autoconf` versions starting from 2.69 instead of 2.70.
- **2026-01-15-fix-sign-normalization-on-MOVE**: **[BUGFIX]** This extension fixes a behavior that prevents sign normalization during a `MOVE` when a `DISPLAY NUMERIC` was not correctly signed during initialization.
- **2026-02-05-add-superbol-testsuite**: This extension introduces a new test suite specific to SuperBOL.
- **2026-02-05-shared-superbol-config**: This extension introduces a common configuration for all options introduced by SuperBOL extensions.
- **2026-02-06-superbol-atscript-tests**: This extension introduces a new format based on `autofonce` for certain SuperBOL regression tests.

- **2026-02-07-allow-more-extra-dots**: **[BUGFIX]** This extension fixes an incorrect GnuCOBOL behavior that does not support redundant periods if they are separated by line directives, typically introduced by `COPY` statements.
- **2026-02-08-allow-unterminated-strings**: **[MF-COMPAT]** This extension allows the absence of a string terminator when the new configuration option `unterminated-string` is enabled.
- **2026-02-08-missing-dot-after-SPECIAL-NAMES**: **[MF-COMPAT]** This extension allows the absence of a period at the end of instructions in `SPECIAL-NAMES`, just before `INPUT-OUTPUT`, `DATA DIVISION`, or `PROCEDURE DIVISION`.
- **2026-02-08-missing-dot-before-or-after-FD**: **[MF-COMPAT]** This extension allows the absence of a period in a field before or after `FD` or `SD`.
- **2026-02-17-autostop**: **[Ergonomics]** This extension allows a COBOL command to be interrupted mid-batch using an environment variable to launch a debugging session. See the Batch Switch section.
- **2026-03-10-handle-spzero**: **[MF-COMPAT]** Compatibility support for the Micro-Focus compiler's `SPZERO` option.

4 Batch Interrupter

Feature: `2026-02-17-autostop`

4.1 Overview

The batch switch allows you to suspend the launch of a specific program within a chain without user intervention. When a program is suspended in this manner, a file containing its PID (process identifier) is generated. Using this information, a debugger such as **gdb** or the one included in **SuperBOL Studio** can be attached to the program to analyze its behavior.

4.2 Prerequisites

Programs to be interrupted and analyzed must be compiled in debug mode (the `-g` option) using a modified GnuCOBOL compiler. The intermediate C files generated by GnuCOBOL must be accessible to the debugger.

4.3 Operation

Prior to launching the chain, the list of programs to be interrupted must be defined in the `COB_STOP_JOBS` environment variable. These can be program names as specified by a `PROGRAM-ID` clause, or executable filenames (for programs compiled as executables using the GnuCOBOL `-x` option). Multiple program names must be separated by commas.

The name of the file used to store the interrupted program's PID can be customized using the `COB_JOB_PIDFILE` variable. If this variable is not set, the file will be named `$EXECUTABLE-$JOB.pid`, where `$EXECUTABLE` is the name of the executable file (`cobcrun` for a module) and `$JOB` is the name of the program as it appears in the `PROGRAM-ID` clause.

The chain can then be executed normally. When it is interrupted, the **gdb** debugger can be attached to the program using the following command:

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

Or, if you prefer to use **gdbserver** instead of gdb:

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

At the start of the debugging session, the program is stopped within a call to the `raise` function of the standard C library. To step back up into the COBOL program (or at least its C translation), you can use the `finish` command (press the Enter key several times to repeat the command).

The debugger included in **SuperBOL Studio** can also be attached to the program by entering the corresponding PID (refer to the SuperBOL Studio documentation for details).

4.4 Example

Assume we have a file `prog.cob` containing two programs with the `PROGRAM-ID`s `prog1` and `prog2`, and a file `subprog.cob` containing a program with the `PROGRAM-ID` `subprog`. The program `prog1` calls `prog2` and `subprog` via a `CALL` statement.

Compile the programs as follows:

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

This produces two binary files: `prog` and `subprog.so`. The `-g` option integrates debugging information into these two binaries and preserves the intermediate C files.

To interrupt the calls to `prog1`, `prog2`, and `subprog`, configure the `COB_STOP_JOBS` environment variable:

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

Note: Since all programs contained within the `prog` executable are targeted, you could also have written:

```
$ export COB_STOP_JOBS=prog,subprog
```

Now, launch the program:

```
$ ./prog
```

Execution is silently interrupted. Using another terminal, locate the file containing the process PID:

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

The execution successfully stopped at program `prog1`. Attach the debugger:

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

The debugger displays:

```
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: No such file or directory.
(gdb)
```

Using the `finish` command, step back up to the program code:

```
(gdb) finish        (press Enter several times)
```

You will see:

```
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)
```

From this point, you can explore the execution of the `prog1` program.

If you resume execution (the `continue` command in gdb), it will automatically stop again at `prog2`, then `subprog`, producing the files `prog-prog2.pid` and `prog-subprog.pid`. (The PID obviously remains the same, but the presence of the file indicates which program has been interrupted).

Note: If `prog.cob` had been compiled as a module rather than an executable, the generated PID files would be named `cobcrun-prog1.pid`, `cobcrun-prog2.pid`, and `cobcrun-subprog.pid`.