SuperBOL Studio User Manual

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 4
2 Introduction 5
3 Installation 6
3.1 Installation within VSCode 6
3.2 Installation via a downloaded VSIX file 7
3.3 SuperBOL Studio OSS vs SuperBOL Studio Pack 10
3.4 Links to the Extensions Pages 10
4 Workspace Configuration 11
4.1 Editing an Existing Project as a Workspace 11
4.2 Configuring the Extension for the COBOL Project 14
4.3 Syntax diagnostics 20
4.4 Collaborating with other developers 20
5 Navigation Features 22
5.1 Outline & Breadcrumbs 22
5.2 Go to Symbol 22
5.3 Go to Definition 23
5.4 Peek Definition 24
5.5 Go to References 25
5.6 Reference Information 26
5.7 Hover to Show Copybooks 27
5.8 Hover to Show Source Text Replacements 28
6 Edition Features 30
6.1 IntelliSense (auto-completion) 30
6.2 Rename Data Items, Sections, and Paragraphs 32
7 Control-Flow Visualization 34
7.1 Exploring the Control-flow 34
7.2 CFG as an arc diagram 35
8 Building Programs 37
8.1 Simple Build 37
8.2 Settings that can change the execution of build tasks 38
8.3 Configuring a build task 38
8.4 Building with a custom configuration 40
8.5 Selecting a default build task 41
8.6 Debugging your programs 41
8.7 Examples of custom configurations 41

8.7.1 Building GnuCOBOL module with debugging informations 42
8.7.2 Building GnuCOBOL executable to check coverage 42
8.7.3 Sending extra arguments to `cobc` 42
9 Debugging Programs 44
9.1 Debugging 44
9.2 Launching the Compiled Program for Debugging 44
9.3 SuperBOL debug configurations 45
9.4 Managing configuration inputs 50
9.5 Customizing the debug configurations 51
9.6 Example of debug configurations 53
9.6.1 Debug a GnuCOBOL module, with a custom `cobcrun` path 53
10 Code Coverage 54
10.1 Coverage Information 54
10.2 Highlighting Coverage Information 54
11 Using Vscode in General 56
11.1 Navigating through Warnings and Errors aka Problems 56
11.2 Switching between Light and Dark Themes 57
11.3 The “Do you trust the authors” popup 57
12 Adding a SuperBOL License 60
12.1 SuperBOL Premium Features 60
12.2 Registering a license in VS Code 60
12.2.1 Register via the Startup Notification 60
12.2.2 Register via the SuperBOL Extension Settings 61
12.2.3 Registering the new SuperBOL license 64
13 SuperBOL File Viewer Premium 65
13.1 Prerequisites 65
13.2 Opening a File 65
13.2.1 Step 1: Open the Data File 65
13.2.2 Step 2: Select a COBOL Program or a Copybook 66
13.2.3 Step 3: Disambiguation (if necessary) 67
13.3 Viewer Interface 69
13.4 Data Visualization 70
13.5 Editing a Record 71
13.6 Formatting 72
13.7 Data Navigation 72
13.8 Filtering 73
13.8.1 Column Set Filtering 73
13.8.2 Column Data Filtering 74

13.8.3 Global Search 75
13.9 Tables (OCCURS clause) 75
14 SuperBOL Studio Reference 80
14.1 SuperBOL Studio Configuration 80
14.2 SuperBOL options in `settings.json` 80
14.3 SuperBOL Options in `superbol.toml` 82
14.4 SuperBOL Commands 82
15 Troubleshooting 84

This manual is available:

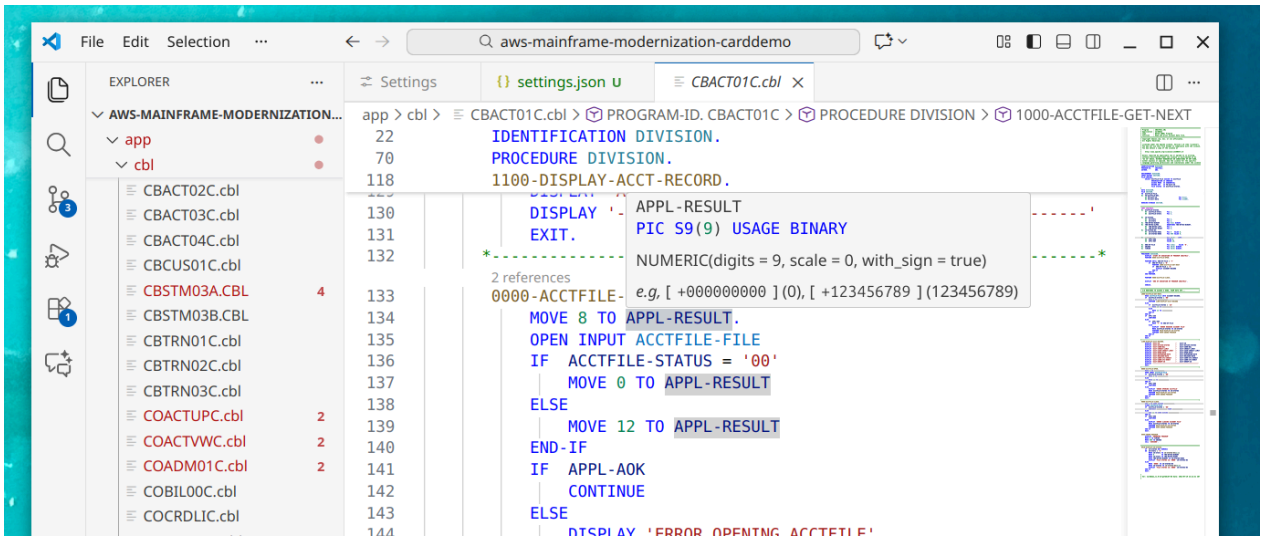
- As an HTML website: in English or in French
- As a PDF File: in English or in French

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

1 Forewords

SuperBOL Studio is an extension for VSCode to develop in COBOL. SuperBOL Studio uses an LSP (Language Server Protocol) Server to understand COBOL sources, display various information and help developers navigate in the sources.

SuperBOL Studio was firstly designed by Titagone-OCamlPro to work with the GnuCOBOL free open-source compiler.



2 Introduction

Welcome to this user manual. This document will explain how to install, configure and use the SuperBOL Studio extension for Vscode to edit COBOL application sources.

The SuperBOL Studio extension allows COBOL developers to benefit from a modern IDE experience to edit their COBOL programs.

Links related with SuperBOL Studio:

- Description on SuperBOL website: https://superbol.eu/solutions/superbol-studio/
- Description on Open-VSX: https://open-vsx.org/extension/OCamlPro/SuperBOL
- Description on Vscode Marketplace: https://marketplace.visualstudio.com/items?itemName=OCamlPro.SuperBOL
- Sources on Github: https://github.com/ocamlpro/superbol-studio-oss
- Another Documentation: https://ocamlpro.github.io/superbol-studio-oss/sphinx

In this manual, screenshots are often taken in a light color theme, to ease printing this document, though many users prefer to use VSCode in the default dark theme. A section in the “Using Vscode” chapter shows how to switch between color themes.

3 Installation

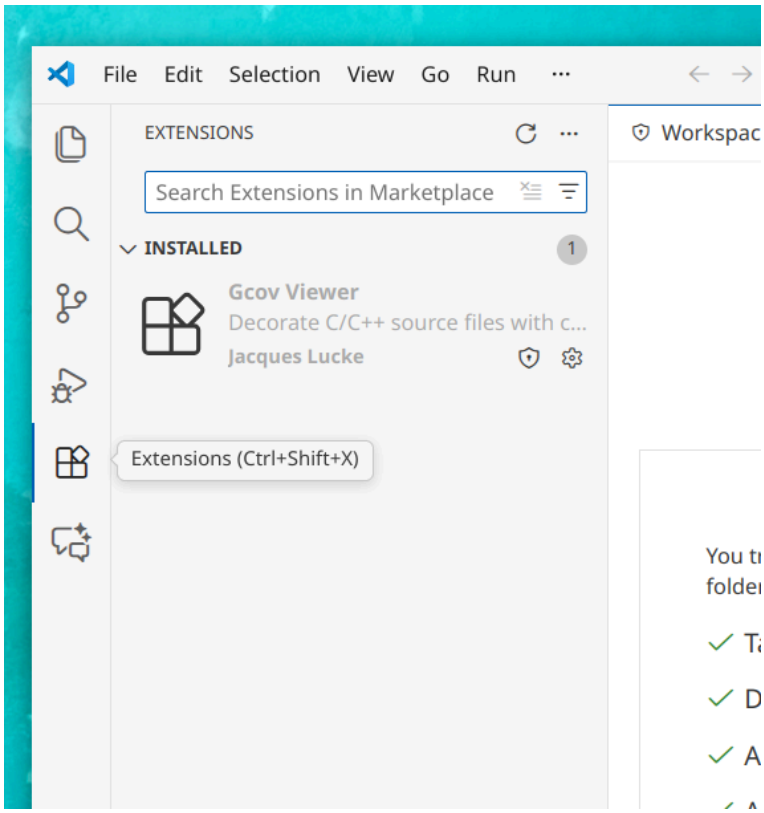
You can install SuperBOL Studio either directly from within VSCode (using the extensions marketplaces), or via a downloaded VSIX file.

3.1 Installation within VSCode

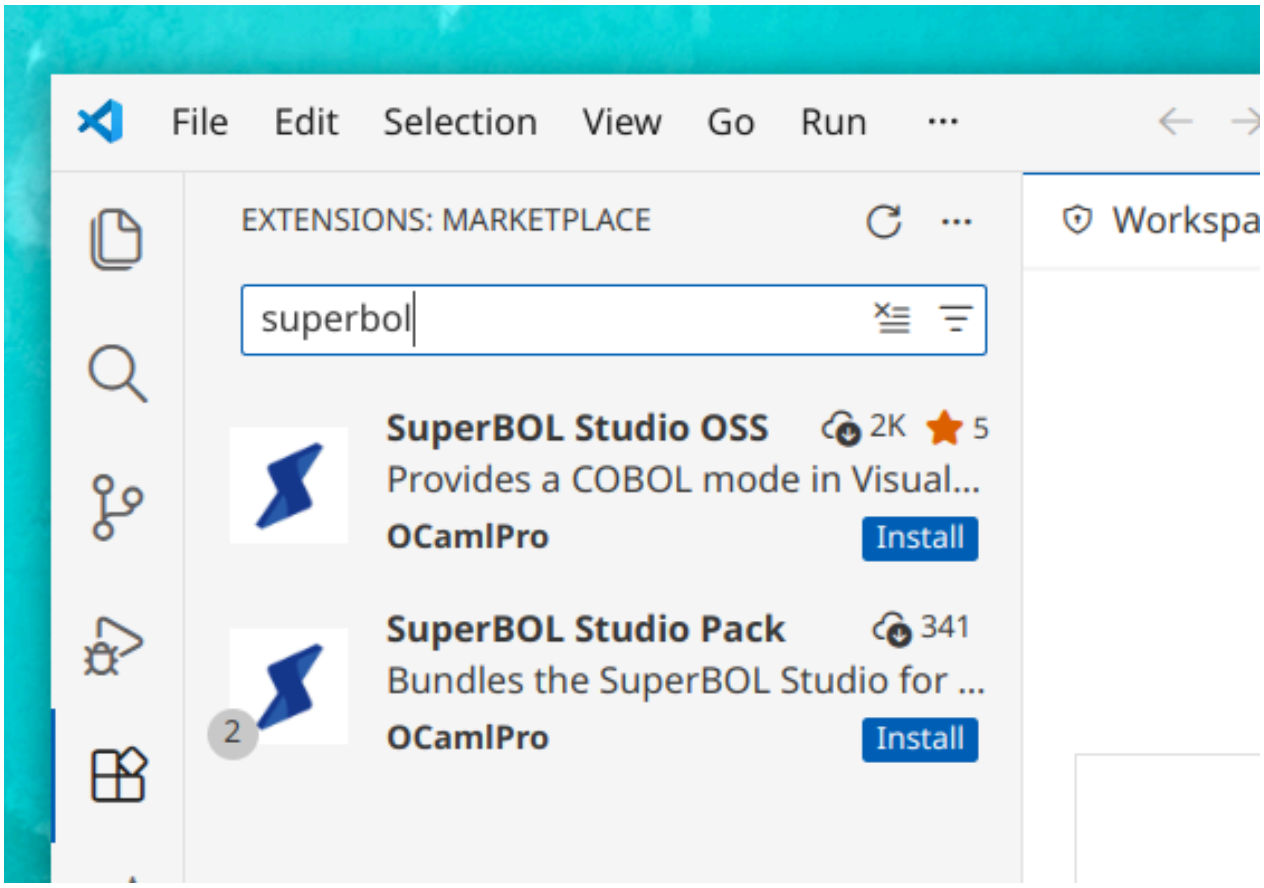
This is the easiest way to install SuperBOL Studio, directly from within VSCode. With this way, you will install the official versions of SuperBOL Studio only. Depending on your version of VSCode, the extensions come either from Microsoft VSCode Marketplace or from the Open-OSX repository, managed by Eclipse Foundation.

Follow these steps:

1. Click on the “Extensions” icon in the activity bar on the left-hand side, or press Ctrl+Shift+X.



2. Search for “superbol”



3. The first items that should appear will be:
   - SuperBOL Studio OSS: the extension that you want to install to use SuperBOL Studio
   - SuperBOL Studio Pack: an extension that contains the "SuperBOL Studio OSS" extension, together with other third-party extensions
4. Click on the `Install` button on one of these two extensions
5. The extension will install itself, and the "Install" button should turn into a configuration wheel button

You can find further instructions for installing extensions directly within VSCode on this page.

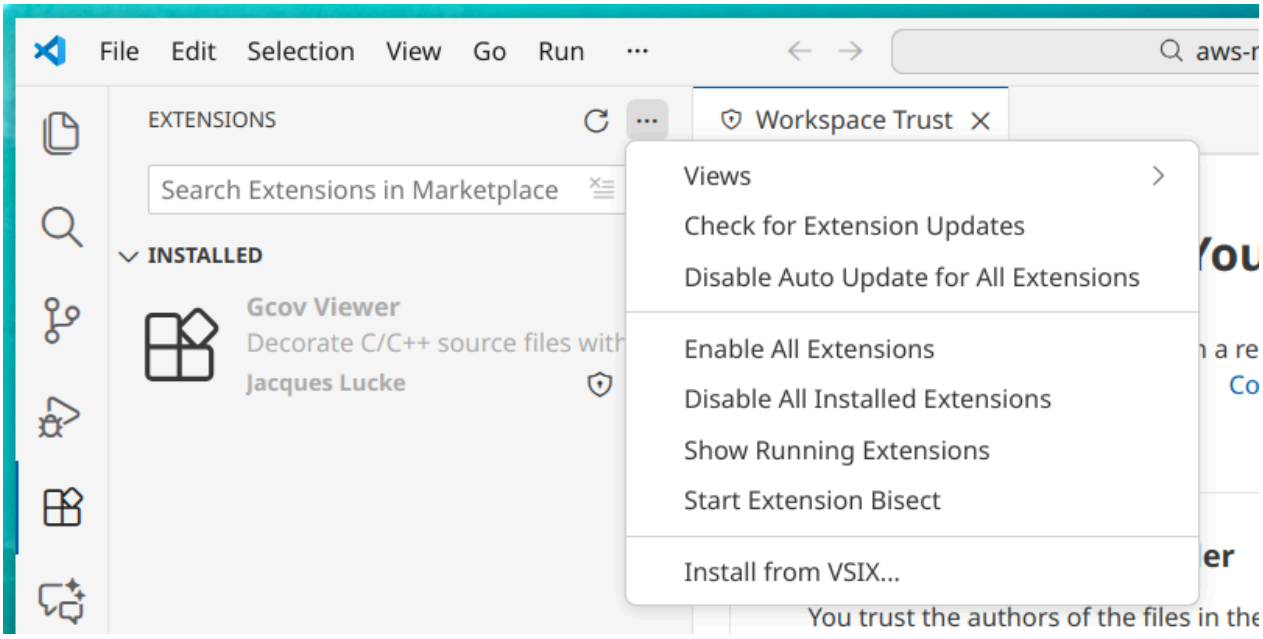
3.2 Installation via a downloaded VSIX file

This is the way to use if you want to install an unofficial version of SuperBOL Studio, typically a test version with beta features, or a version compiled from the sources on Github.

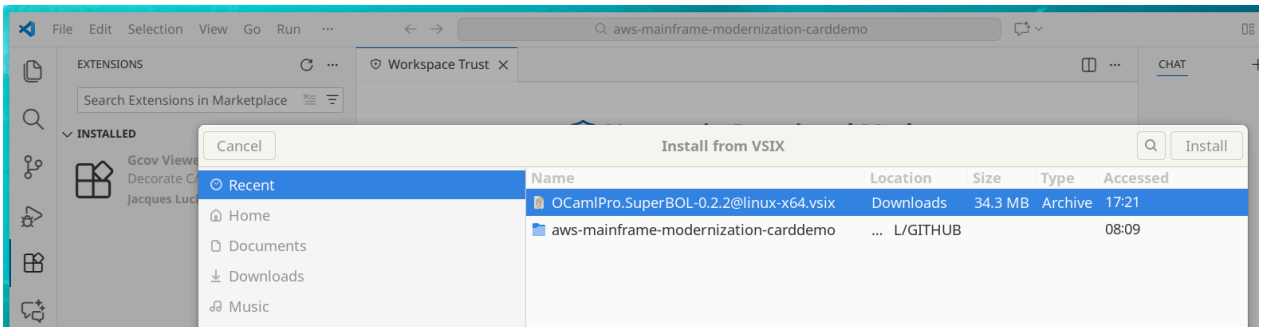
Follow these steps:

1. Check the location of the extension file on your computer. It should come as a single file with the `.vsix` extension
2. In VSCode, go to the the "Extensions" view

3. In the sidebar, click on the three dots (`...`) on the top right-hand side (just above `search`),



4. Select `Install from VSIX...`, and a dialog box should appear
5. Pick the VSIX file on your disk, and click Install



Note that you can also use this method to install Vsix files downloaded from the marketplaces. For that, you need to select the correct binary for your platform:

Works With

VS Code: ^1.64.0

Target Platforms: Windows x64, Linux x64, macOS Intel, Universal

Resources

Homepage

Repository

Bugs

DOWNLOAD

- Windows x64
- Linux x64
- macOS Intel
- Universal

ad, you accept this
Use.

s

cov-viewer

Report Abuse

3.3 SuperBOL Studio OSS vs SuperBOL Studio Pack

The SuperBOL Studio OSS extension contains the SuperBOL Studio basic features. OSS stands for "Open-Source Software", as the extension is open-source: most of the sources are publicly available on Github (allowing to rebuild the .vsix file with public features), and the other sources are provided to SuperBOL's customers within the "Subscription" contract.

The SuperBOL Studio Pack extension is a "meta extension": it selects a set of extensions to be installed together. This is usually useful if you want to benefit from our choice of some other extensions that we found useful to use together with SuperBOL Studio OSS.

Currently, SuperBOL Studio Pack includes:

- SuperBOL Studio OSS, of course
- tamasfe.even-better-toml: a mode to edit TOML files, that can typically be used to edit the `superbol.toml` file for advanced configuration. See https://marketplace.visualstudio.com/items?itemName=tamasfe.even-better-toml

3.4 Links to the Extensions Pages

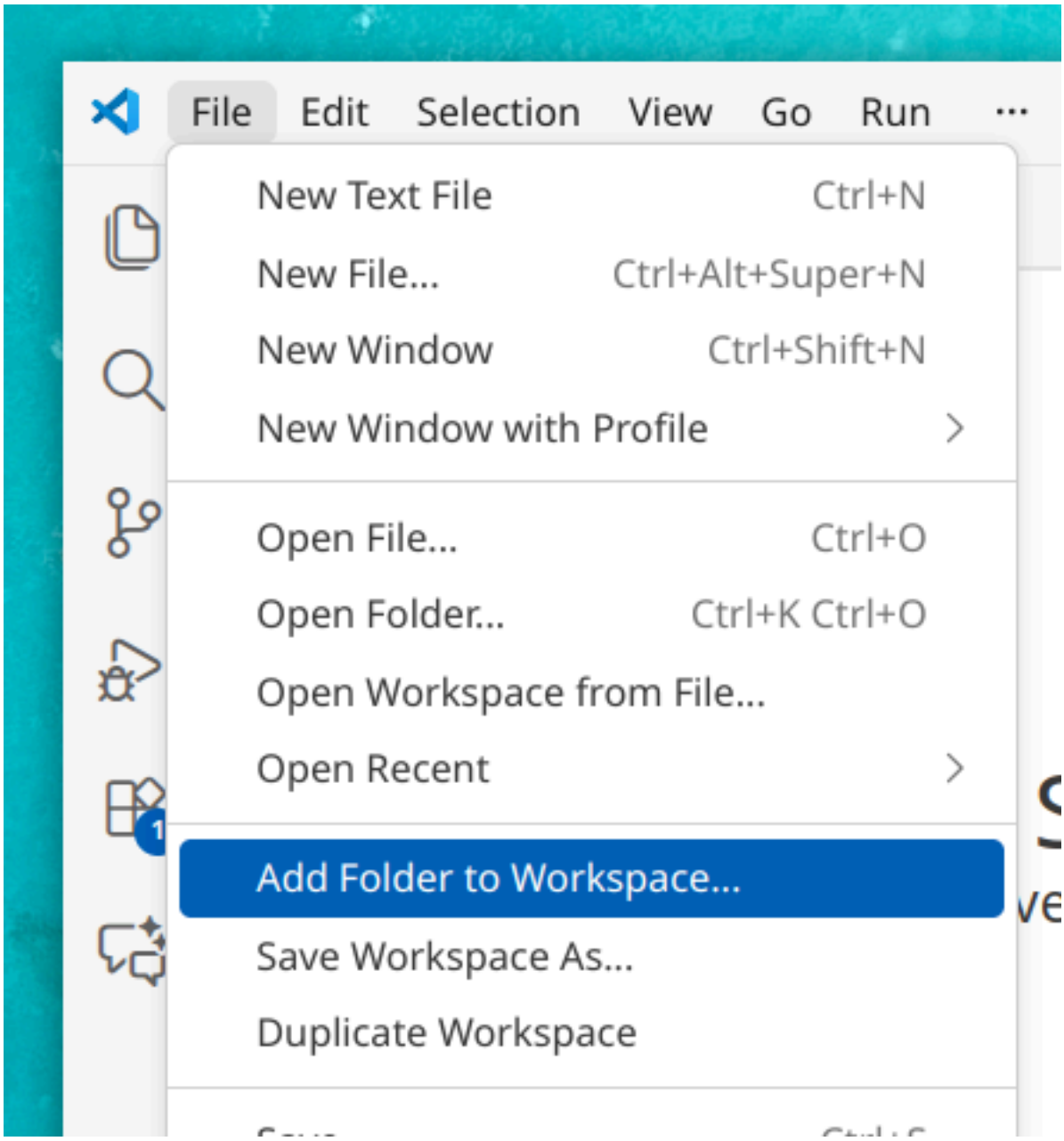
- Descriptions on Open-VSX: https://open-vsx.org/
  - SuperBOL Studio OSS: https://open-vsx.org/extension/OCamlPro/SuperBOL
  - SuperBOL Studio Pack: https://open-vsx.org/extension/OCamlPro/SuperBOL-studio-pack
- Descriptions on Vscode Marketplace:
  - SuperBOL Studio OSS: https://marketplace.visualstudio.com/items?itemName=OCamlPro.SuperBOL
  - SuperBOL Studio Pack: https://marketplace.visualstudio.com/items?itemName=OCamlPro.SuperBOL-studio-pack
- Sources on Github: https://github.com/ocamlpro/superbol-studio-oss

4 Workspace Configuration

SuperBOL Studio can be used both on single COBOL files, or on a full COBOL project, usually called a Workspace in VSCode language.

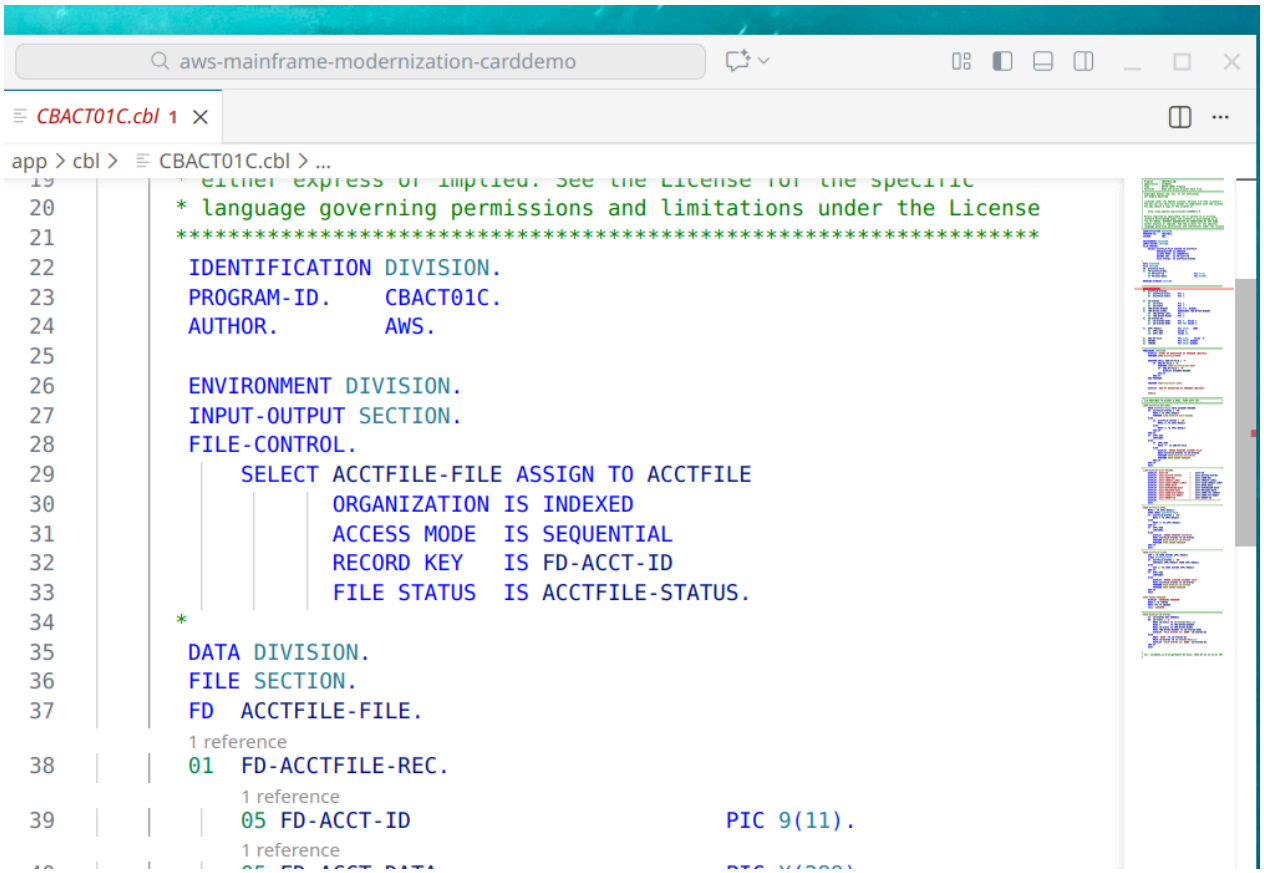
4.1 Editing an Existing Project as a Workspace

To start using the extension on an existing project, open its folder in VSCode (`File > Add Folder to Workspace…`).



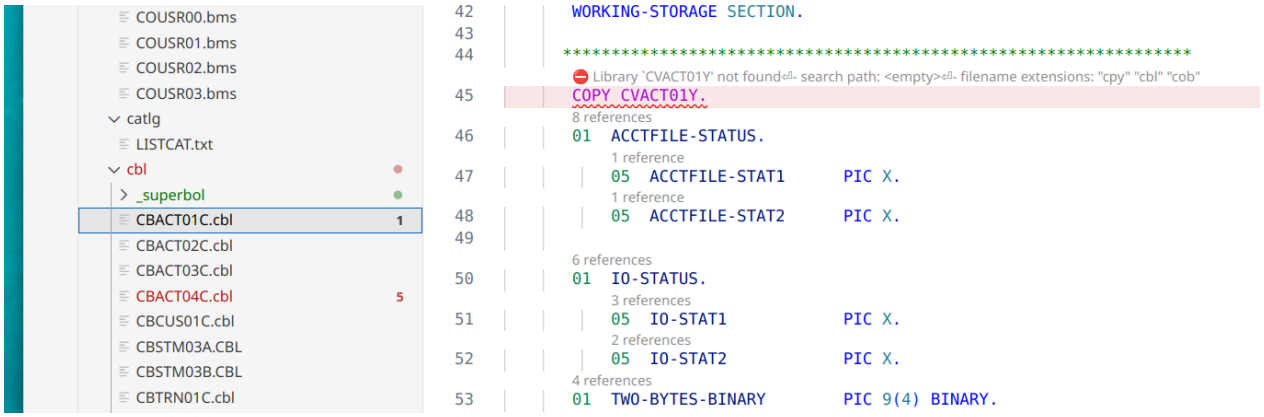
The extension will start automatically whenever the folder contains files with usual COBOL filename extensions (`.cob`, `.cbl`, `.cpy`, `.cbx`).

Once the extension is started, COBOL sources should appear with colorization:



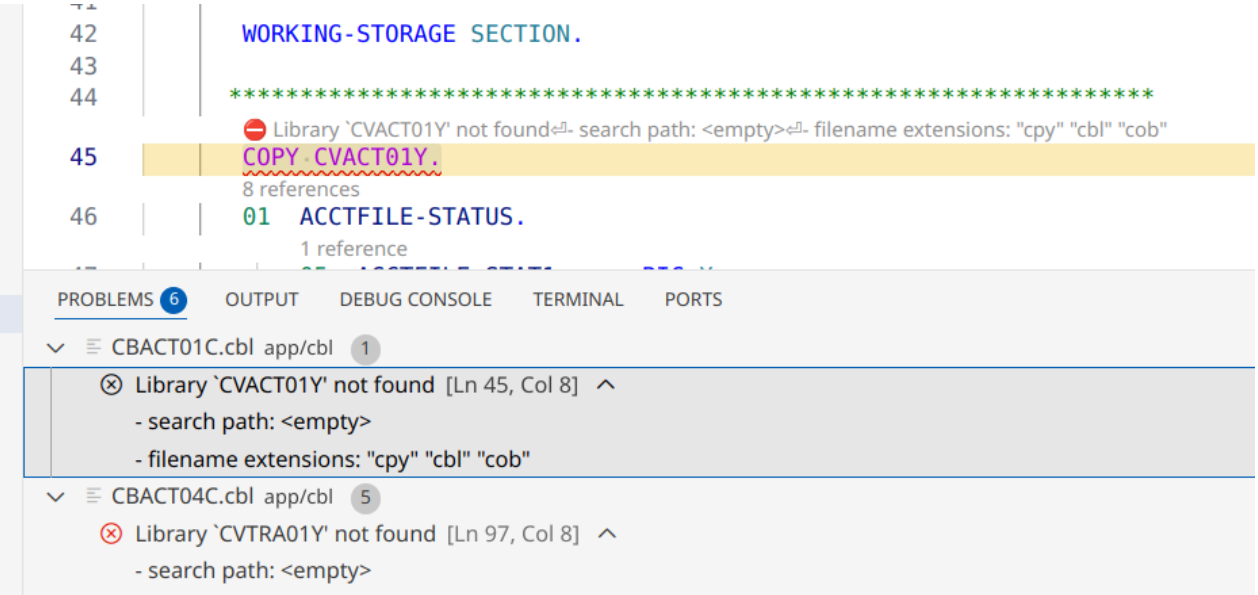
The extension will try to parse the sources, and resolve COPY statements. If it fails, it will display problems:

- the number of problems found after the file name in the file explorer
- these problems are shown directly in the sources



Note that, in our case, we use the "Error Lens" VSCode extension to inline errors in the source code. There are many similar extensions to improve the visualization of problems.

You can also directly go to the first problem, using Ctrl+Shift+M:



Such problems usually come from a bad configuration of the workspace for the COBOL dialect used by this project. So we need to configure it to make the extension work correctly. We will see this in the next section.

4.2 Configuring the Extension for the COBOL Project

Open the settings (`File > Preferences > Settings`, or Ctrl+,), and start typing “superbol...”. You will be presented with a screen that resembles:

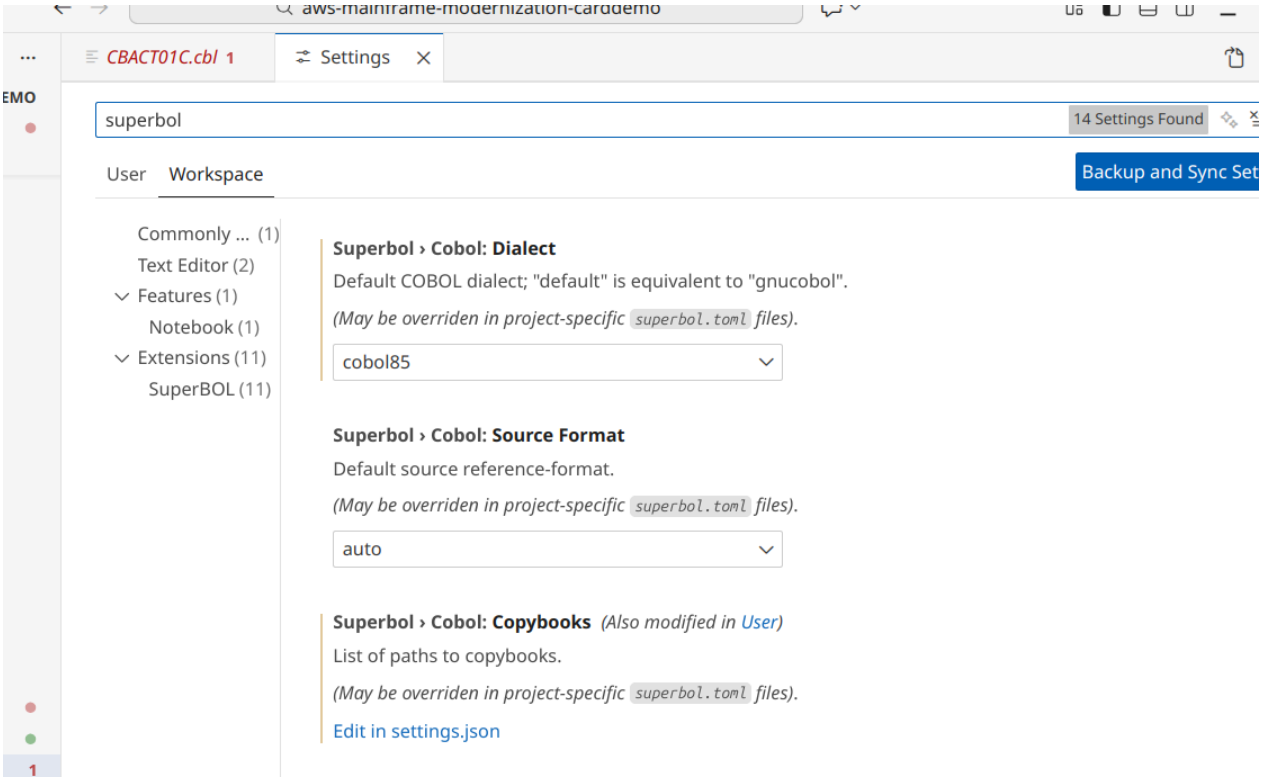


Figure 1: SuperBOL settings

Beware that there are several ways to configure settings:

- **User** settings: these are the default settings when the workspace is not configured. On the picture above, they are on the left tab “User”. They are stored in the user profile, typically in `$HOME/.config/Code/User/settings.json`.
- **Workspace** settings: these are the settings specific to this workspace. **They are the ones you probably want to configure now.** On the picture above, they are on the right tab “Workspace”. They are stored in a file `.vscode/settings.json` at the root of the workspace.
- Additionnaly, SuperBOL Studio can use its own configuration file for some settings, located in a file **superbol.toml** at the root of the project. This file is typically useful if you plan to use SuperBOL tools on the command-line in this project. VSCode will sometimes warn you if you are trying to configure settings internally in VSCode when there is also a `superbol.toml` file.

In our example, the source format was correctly inferred. If it is not your case, you can change the **Source Format** option:

- The default reference source-format `"superbol.cobol.sourceFormat"` When `auto` is selected, which is the default, SuperBOL (and GnuCOBOL) will automagically try to guess whether the source is in `free` or `fixed` format. Other source formats need to be configured explicitly.

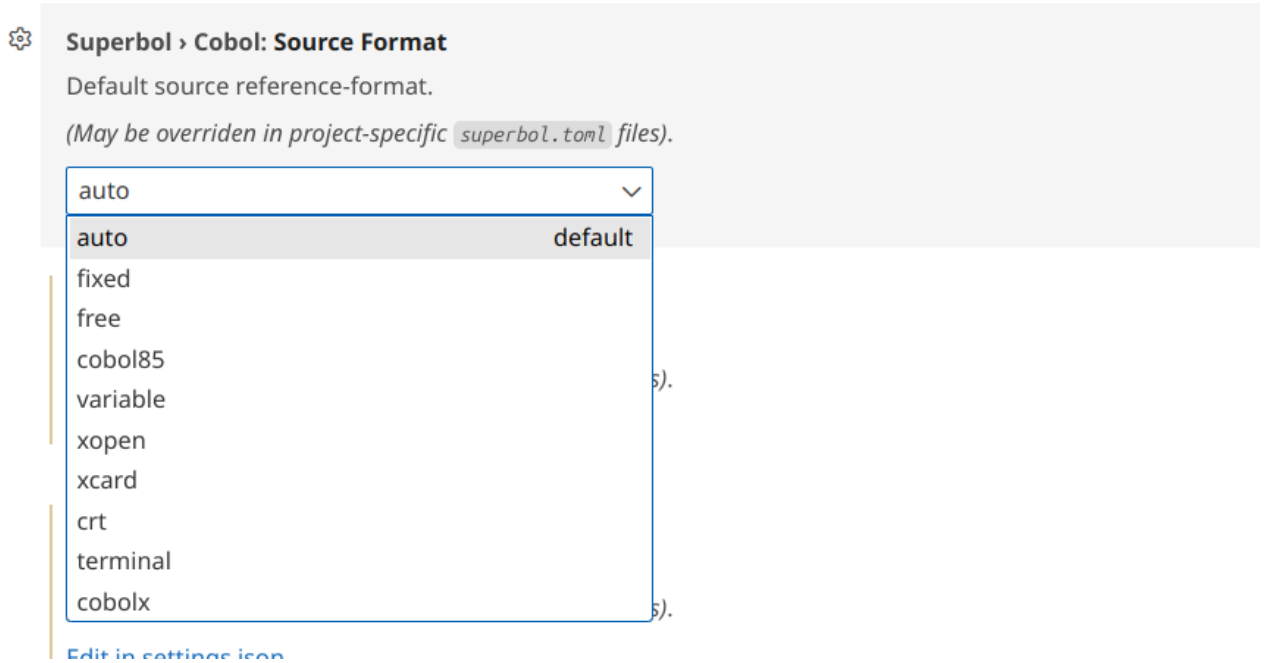
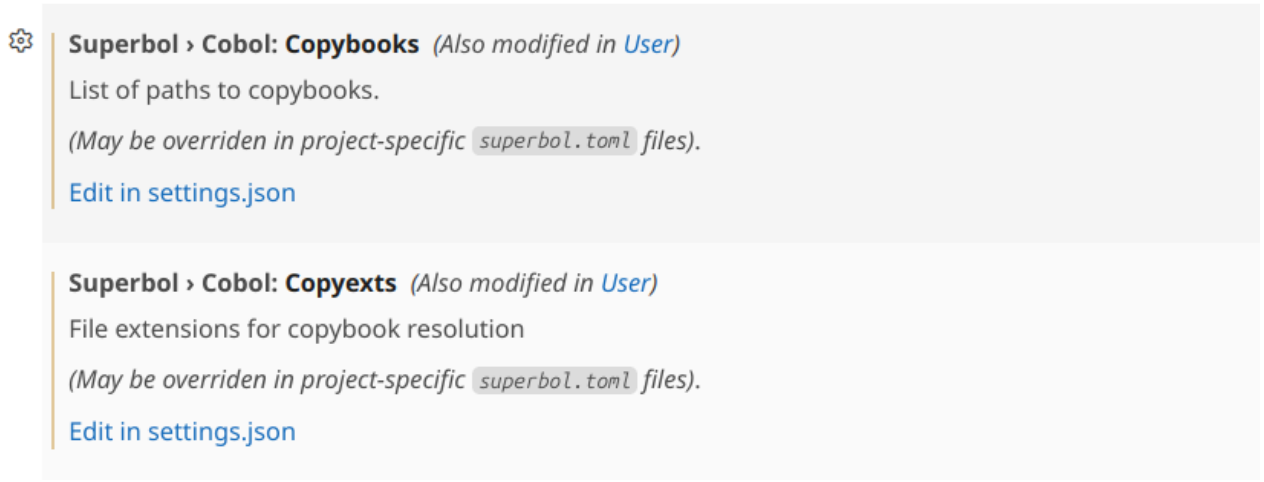


Figure 2: SuperBOL settings

In our case, the extension complains about not finding some copybooks. Two settings are available to control how the extension looks up for copybooks:



Both options have to be configured in JSON by modifying the `settings.json` file, so you need to click on “Edit in settings.json”.

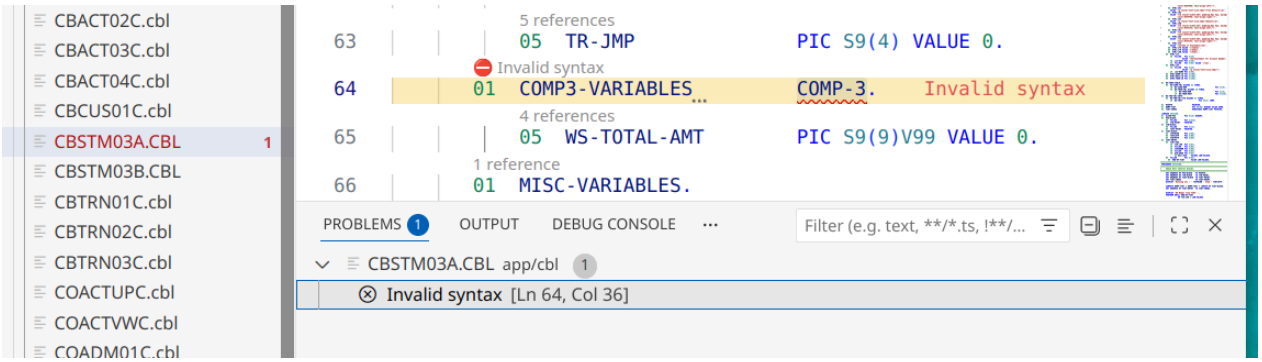


Figure 3: Editing copybook paths in `.vscode/settings.json`

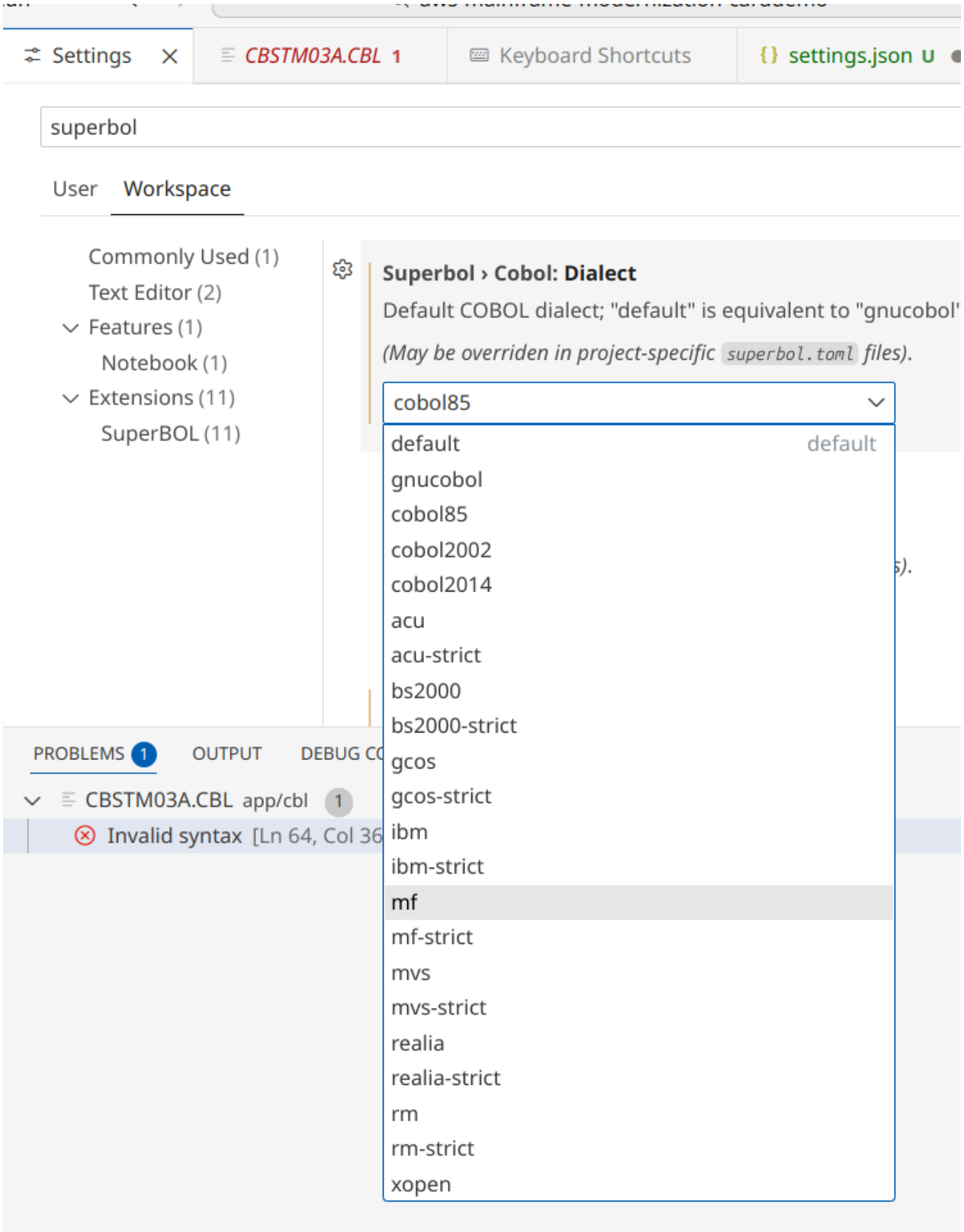
- The path to copybooks **"superbol.cobol.copybooks"**. This is a list, where each entry describes an element of the search path where your copybooks will be looked for. Each entry must contain a field **"dir"**, and an optional field **"file-relative"**.
  - If the directory you want to specify (where copybooks are located) is a path starting from the root of the workspace, or an absolute path, you can skip the `file-relative` field, or set it to `false`.
  - If the directory you want to specify is a path relative to the directory containing the specific COBOL source with the `COPY` statement, you should set the `file-relative` field to `true`.
- The copybook extensions **"superbol.cobol.copyexts"**. To configure this setting, you will need to select `Edit in settings.json`. This is a list, where each entry describes a file extension if the copybook name cannot be found as-is (for example `COPY "mycpy.lib"`). In SuperBOL, the `default` option corresponds to GnuCOBOL's default, which is `["cpy","cbl","cob"]` (searched in upper-case first, and then in lower-case).

Do not forget to save the file (with Ctrl+S). As soon as it is saved, the extension will restart the LSP to re-analyze the COBOL sources.

Now, the extension gives us a different kind of errors, because it couldn't understand the syntax:



- The COBOL dialect used in the project `"superbol.cobol.dialect"`: In SuperBOL, the `default` dialect corresponds to GnuCOBOL's default, that supports many features from dialects such as `COBOL2014`, `IBM`, `MicroFocus` (`mf`), or `GCOS`.


Settings
CBSTM03A.CBL 1
Keyboard Shortcuts
settings.json U
superbol
User
Workspace
Commonly Used (1)
Text Editor (2)
Features (1)
Notebook (1)
Extensions (11)
SuperBOL (11)
Superbol › Cobol: Dialect
Default COBOL dialect; "default" is equivalent to "gnucobol"
(May be overriden in project-specific superbol.toml files).
cobol85
default
default
gnucobol
cobol85
cobol2002
cobol2014
acu
acu-strict
bs2000
bs2000-strict
gcos
gcos-strict
ibm
ibm-strict
mf
mf-strict
mvs
mvs-strict
realia
realia-strict
rm
rm-strict
xopen
PROBLEMS 1
OUTPUT
CBSTM03A.CBL app/cbl 1
Invalid syntax [Ln 64, Col 36

4.3 Syntax diagnostics

> **Note**
>
> Syntax checks performed by SuperBOL Studio currently cover the `COBOL85` dialect, and some constructions of more recent dialects supported by GnuCOBOL. Reporting of such diagnostics is currently disabled for dialects other than `COBOL85` to avoid misleading developers with false diagnostics about syntax errors.
>
> Reporting can be re-enabled for every dialect by setting the `Force Syntax Diagnostics` flag in SuperBOL configuration settings.

4.4 Collaborating with other developers

At this point, the settings for your project are stored and managed by VS Code. However, you may plan to collaborate with developers that do not use this editor. For instance, they might want to use our mode for GNU/Emacs, which is located here. Then, we advise you to let SuperBOL Studio store the configuration in a `superbol.toml` file that will be located at the root of the project.

You can make the extension write your current project configuration into a `superbol.toml` by entering the command palette (`View > Command Palette...`, or press Ctrl+Shift+P), and then selecting the command `SuperBOL: Write Project Configuration`.

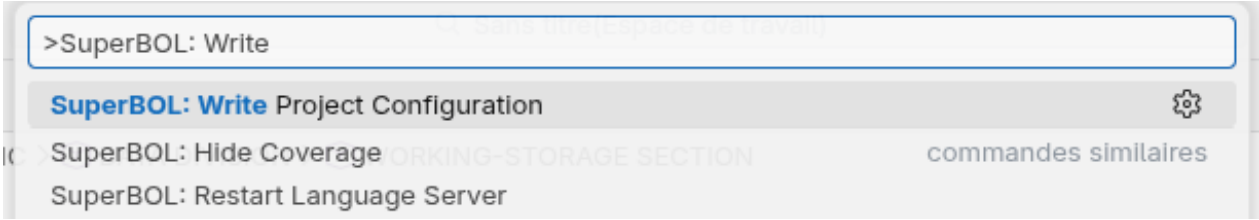


Figure 4: SuperBOL: Write Project Configuration

This will save a `superbol.toml` file at the root of each currently opened project directory. Such a file will not store any user-specific settings, so you can now safely put them under source control. Extensions dedicated to the edition of TOML files, such as `tamasfe.even-better-toml`, provide the same level of assistance as when you edit `.vscode/settings.json`.

> **Tip**
>
> Install the `OCamlPro.SuperBOL-studio-pack` extension to get SuperBOL Studio and `tamasfe.even-better-toml` altogether.

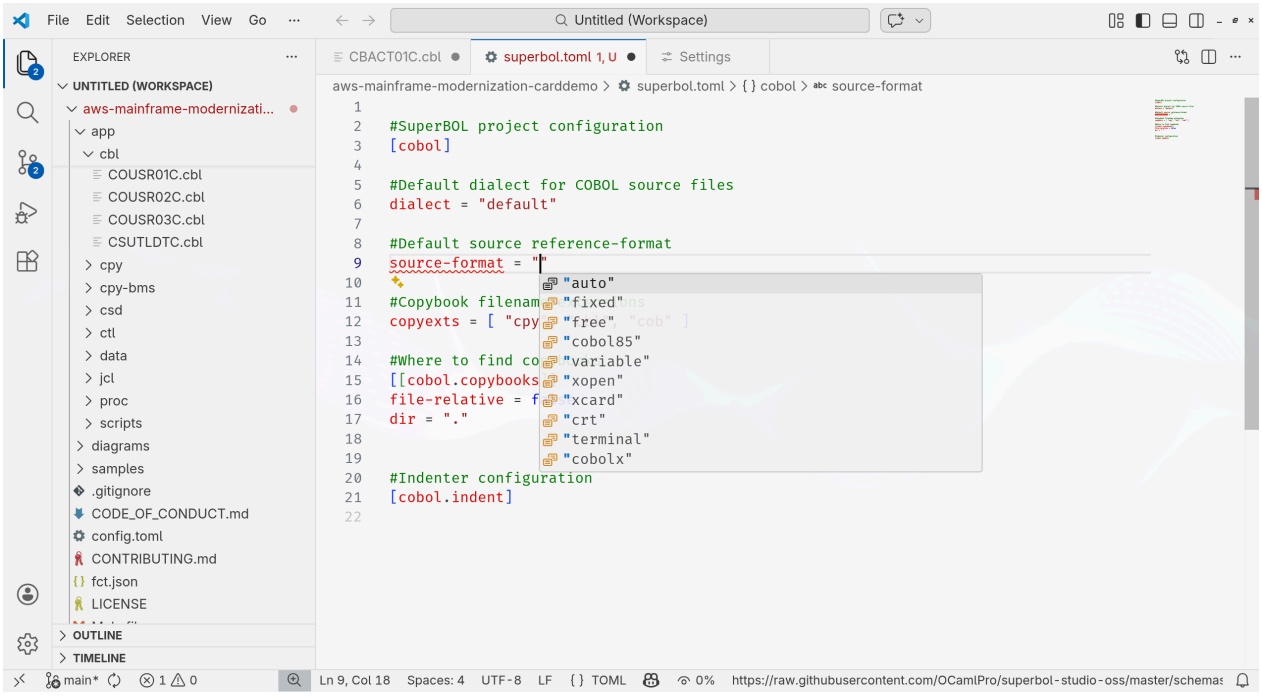


Figure 5: Editing `superbol.toml`

5 Navigation Features

5.1 Outline & Breadcrumbs

SuperBOL provides an outline view of your program once you open it, that you can use to navigate to specific sections or symbols (data items, paragraphs, etc). The same information is also shown in the "breadcrumbs" bar, that is usually located above the text edition area.

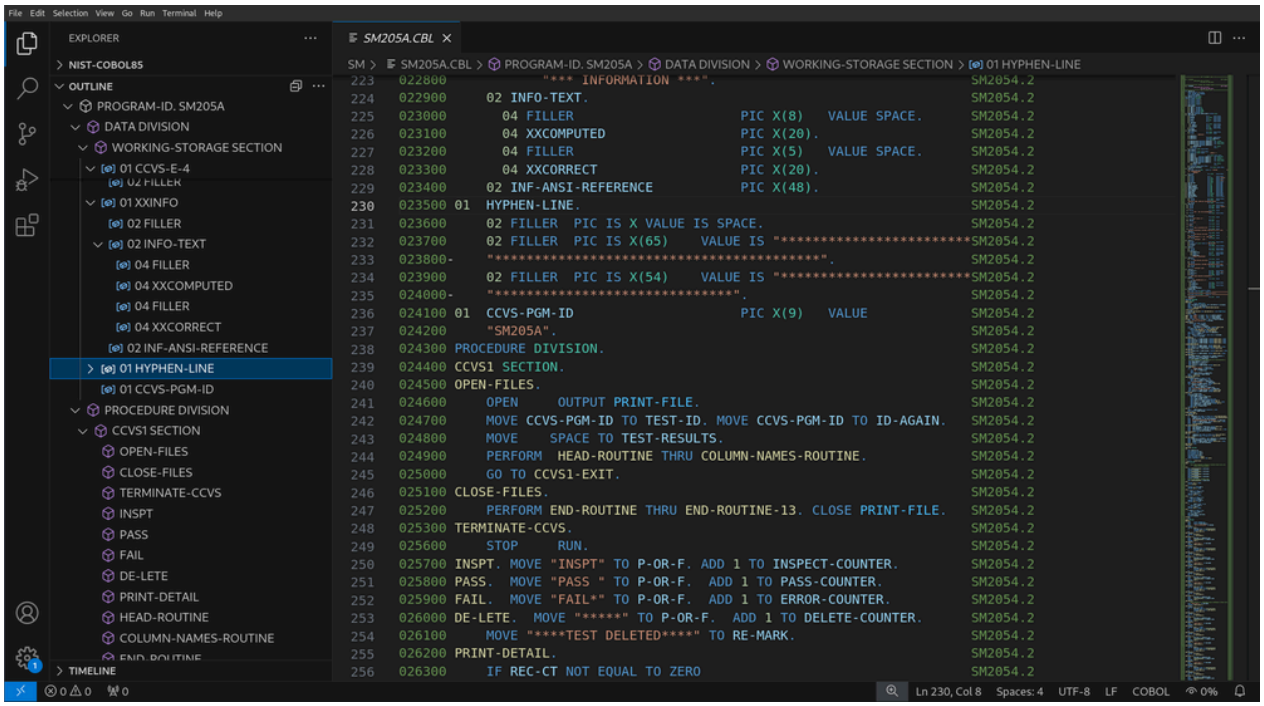


Figure 6: Outline & Breadcrumb

5.2 Go to Symbol

Symbols shown in Outline and Breadcrumbs views can also be searched and jumped to by pressing Ctrl+Shift+O.

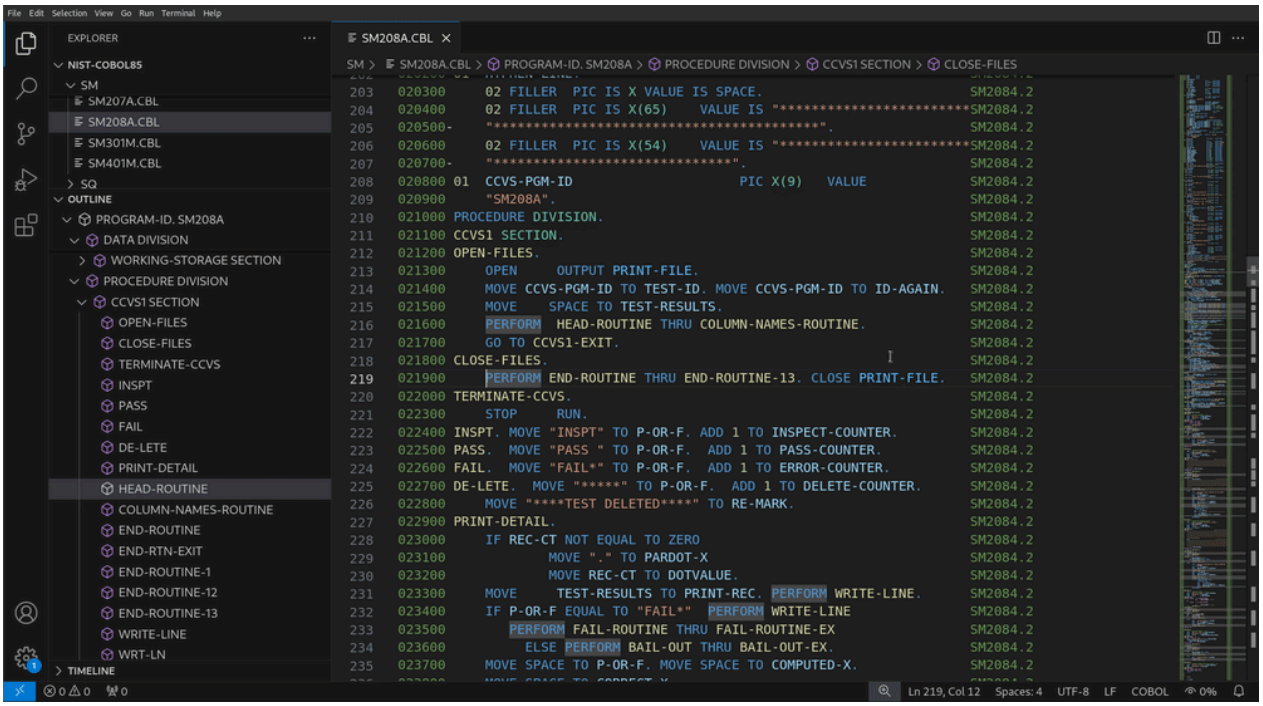


Figure 7: Go to Symbol

5.3 Go to Definition

When you want to locate the definition of a data item name in your source code, position your cursor on its name, right click, and select `Go to Definition` (or press F12).

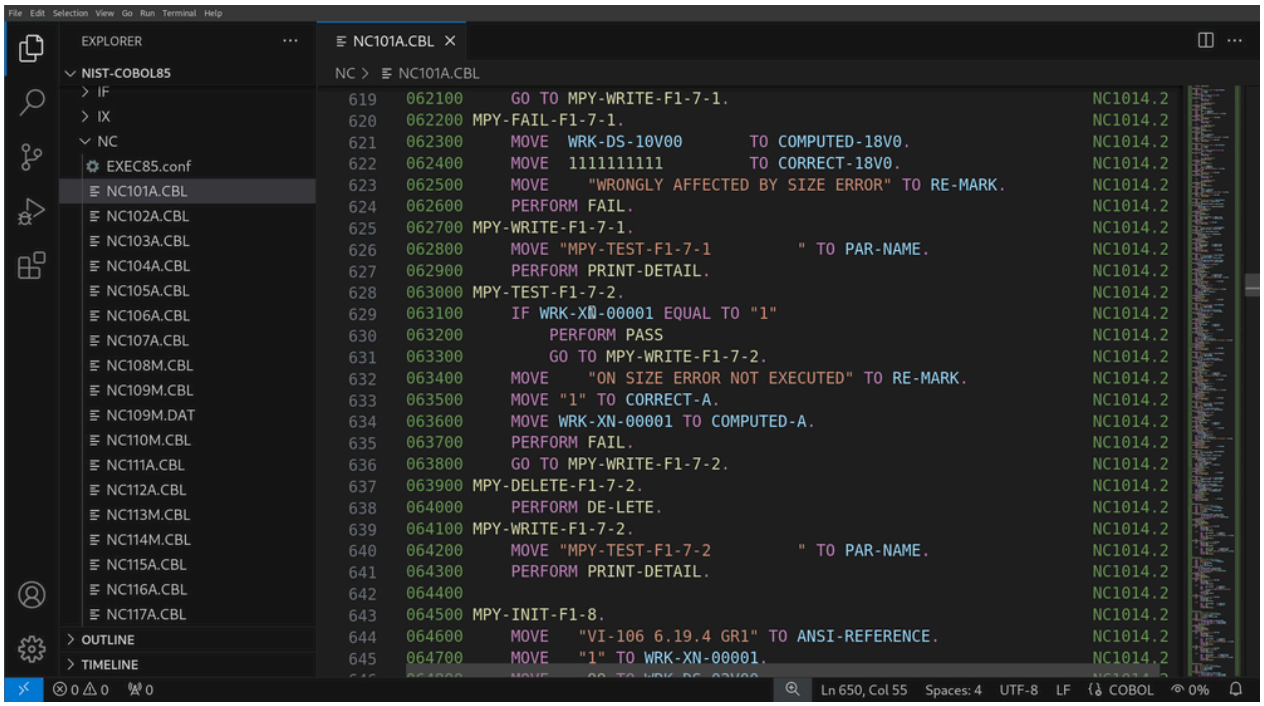


Figure 8: Go to Definition

> **Note**
>
> (Temporary limitation)
>
> At the moment, definitions that belong to communication, report, or screen sections of the data division are ignored by the extension. In addition, some definitions in embedded SQL blocks (`EXEC SQL`) are not taken into account yet.

5.4 Peek Definition

To only have a peek at where such a data item defined, you can position the cursor on its name, right click, and select `Peek > Peek Definition` (or press Ctrl+Shift+F10). You will then be presented with a view of the location of the corresponding definition, including if it lies in a copybook.

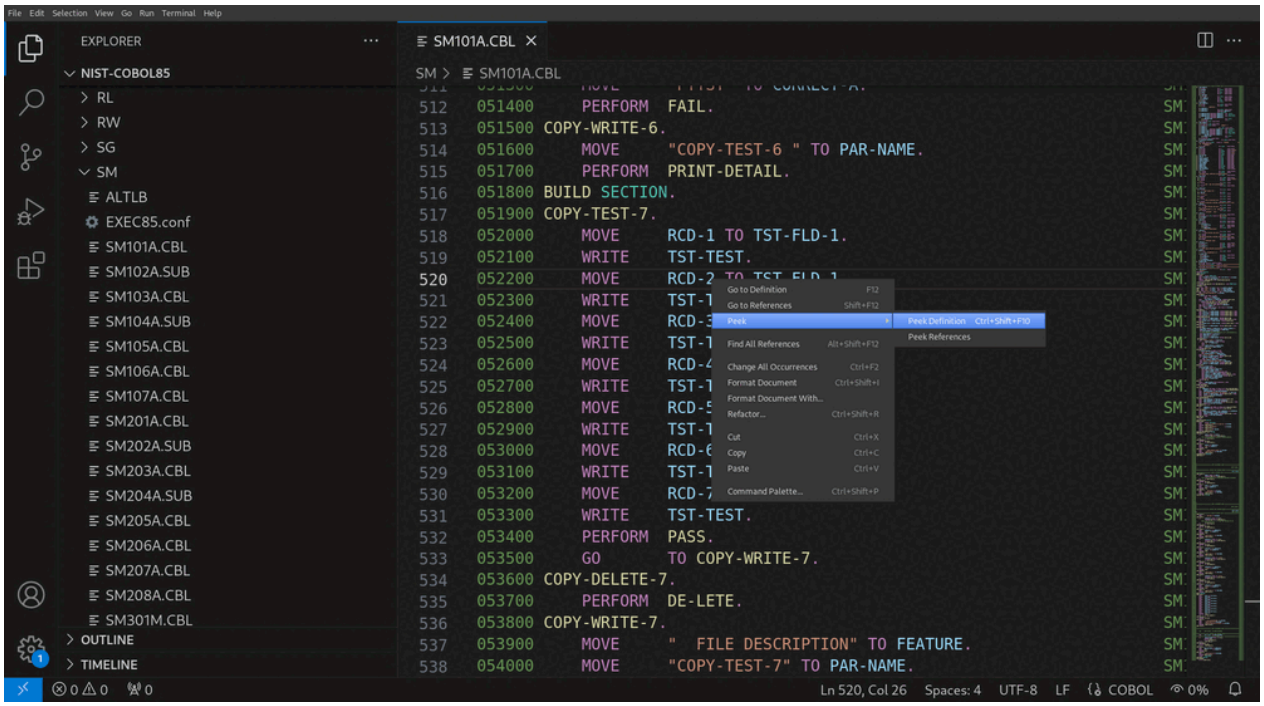


Figure 9: Peek Definition in copybook

5.5 Go to References

If you want to obtain a list of all references to a named data item, right click and select `Go to References` (or press Shift+F12). You will then view the location of every reference to this item.

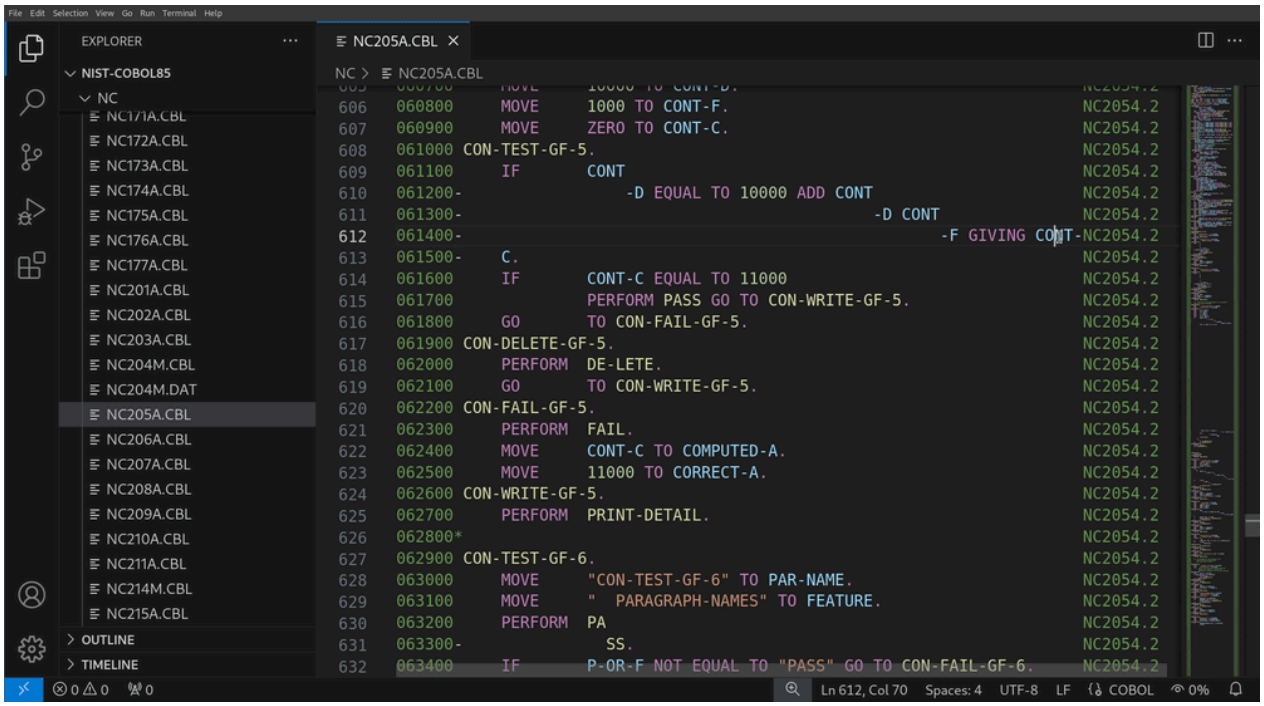


Figure 10: Go to References

> **Note**
>
> (Temporary limitation)
>
> Limitations mentioned in Go to Definition also apply.

5.6 Reference Information

The extension shows inline reference information above definitons of data items and elements of the procedure division.

> **Note**
>
> The same limitations as for Go to Definition apply.

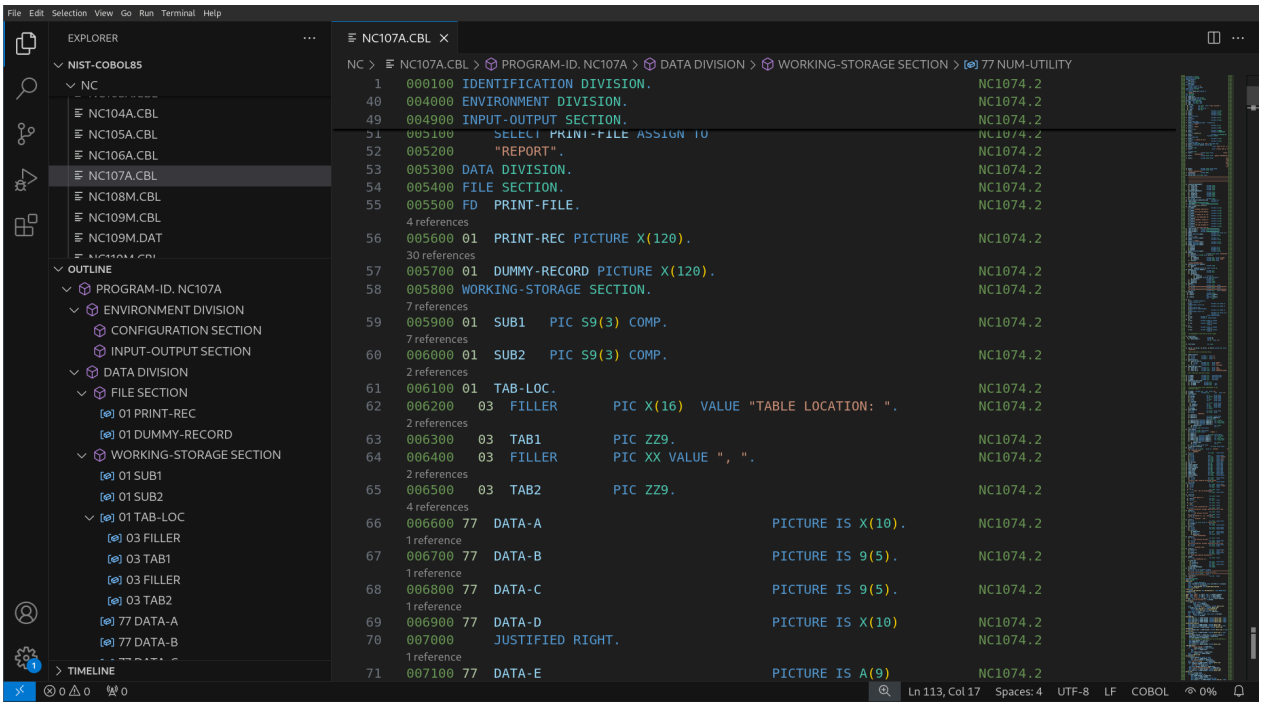


Figure 11: Reference Information

> **Tip**
>
> This feature can be turned on or off by tuning the `"editor.codeLens"` configuration setting (you can type Ctrl+, and then `codelens` to change this setting).

5.7 Hover to Show Copybooks

Ever wondered what was behind a `COPY` directive? Just position your cursor over such a statement, and you will be presented with the contents of the copybook.

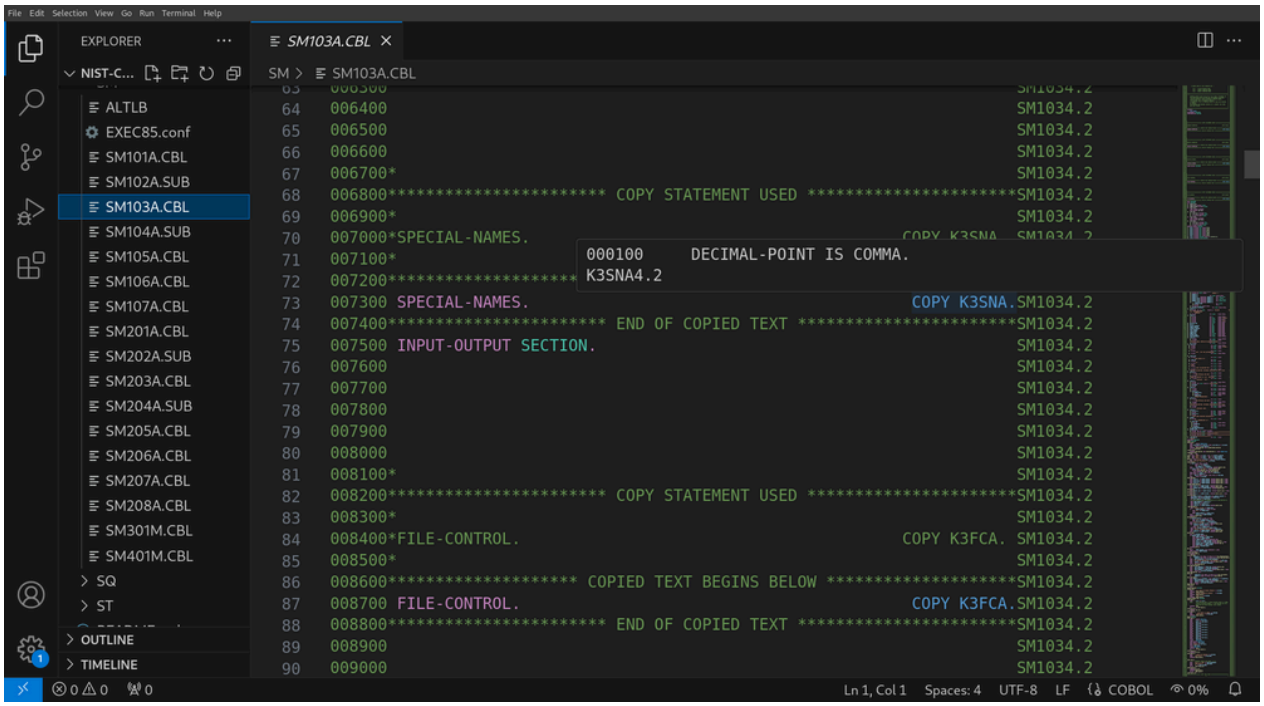


Figure 12: Hover over `COPY`

To ensure this works correctly, check your `"superbol.cobol.copybooks"` and `"superbol.cobol.copyexts"` settings.

5.8 Hover to Show Source Text Replacements

What's more? You can see the source text that results from replacement by a `REPLACE` directive in the same way.

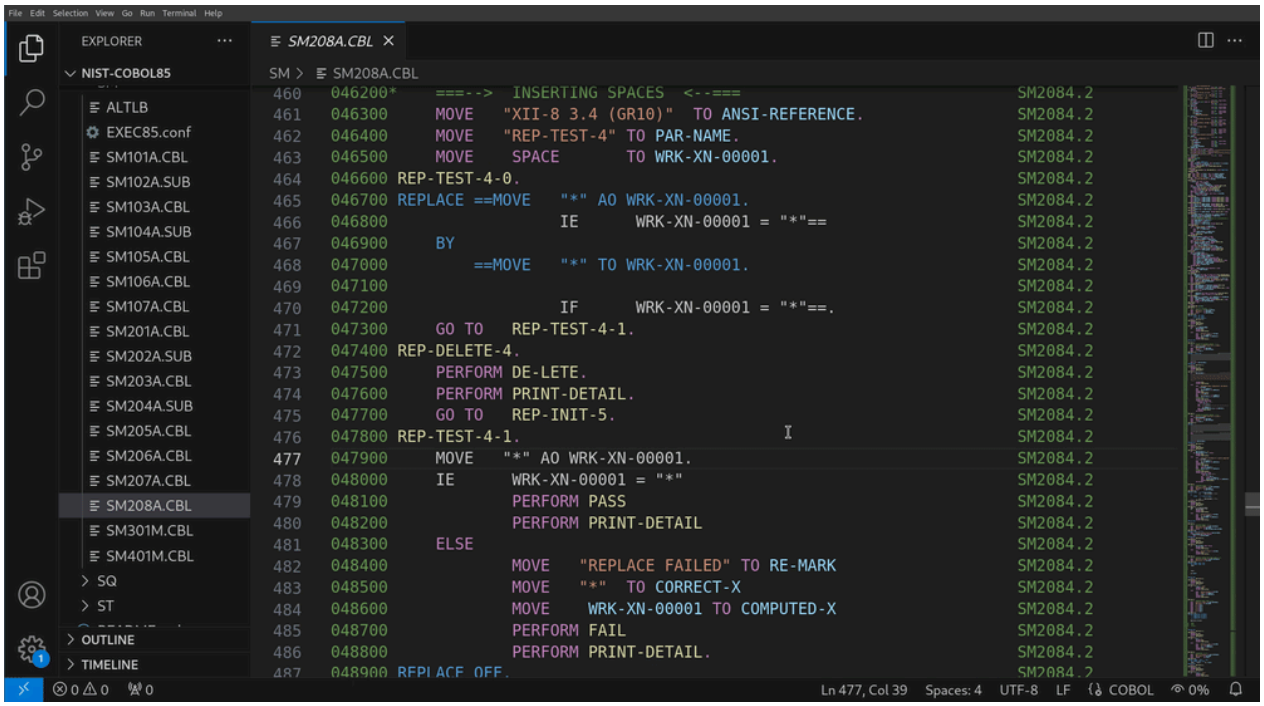


Figure 13: Hover over replacement

6 Edition Features

6.1 IntelliSense (auto-completion)

When editing a program, you can press Ctrl+Space to obtain suggestions on valid keywords, user-defined words (data item or paragraph names), and even complete COBOL sentences. Select an option with the arrow keys, and press Enter to insert the selected suggestion.

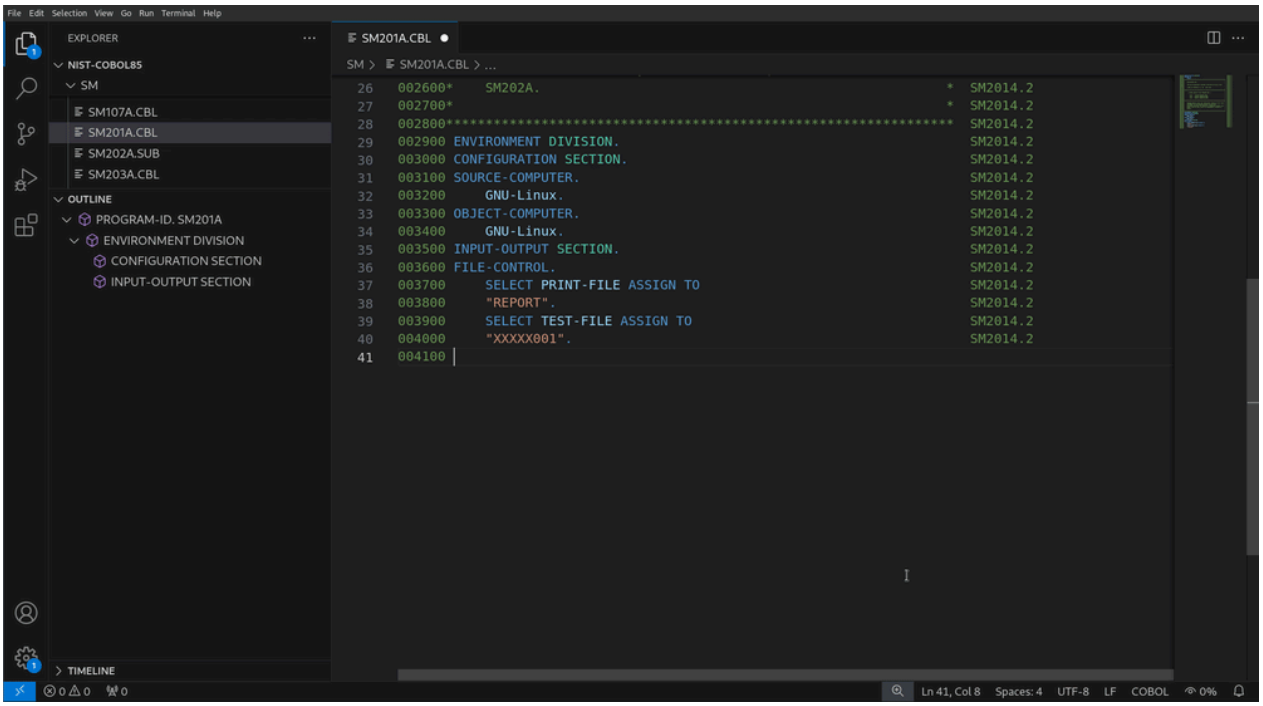


Figure 14: IntelliSense sentence

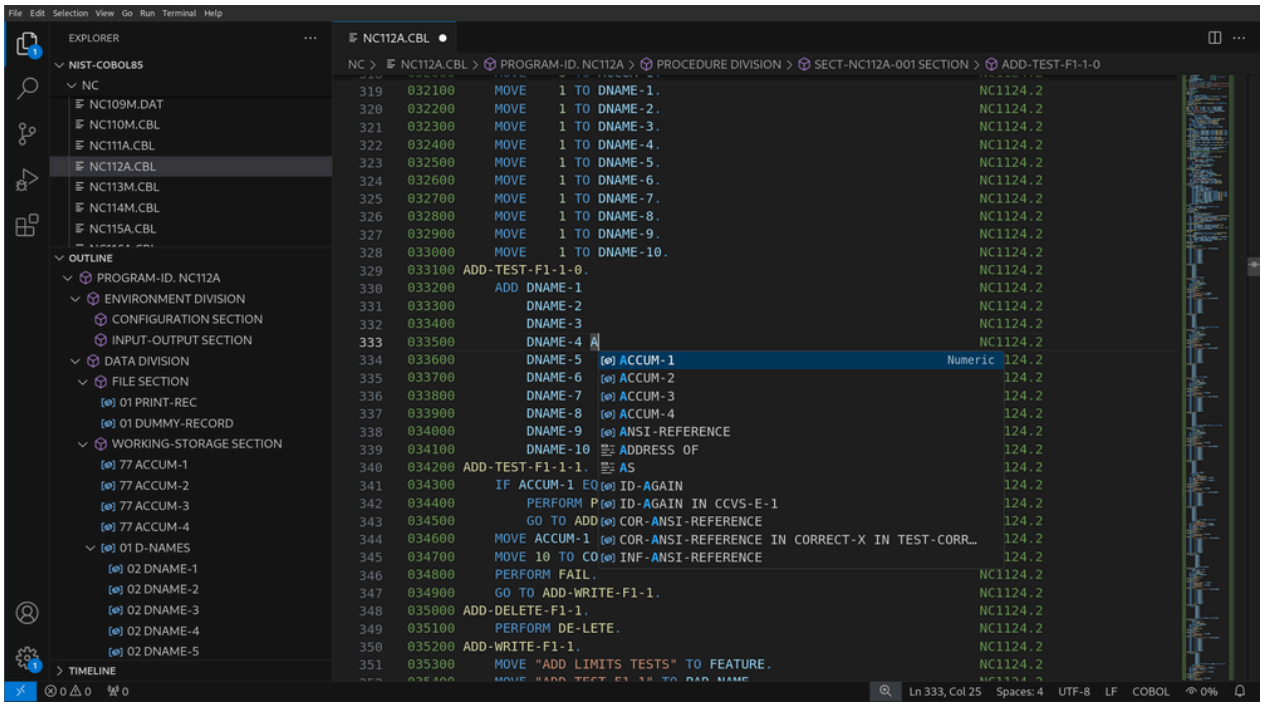


Figure 15: IntelliSense data item

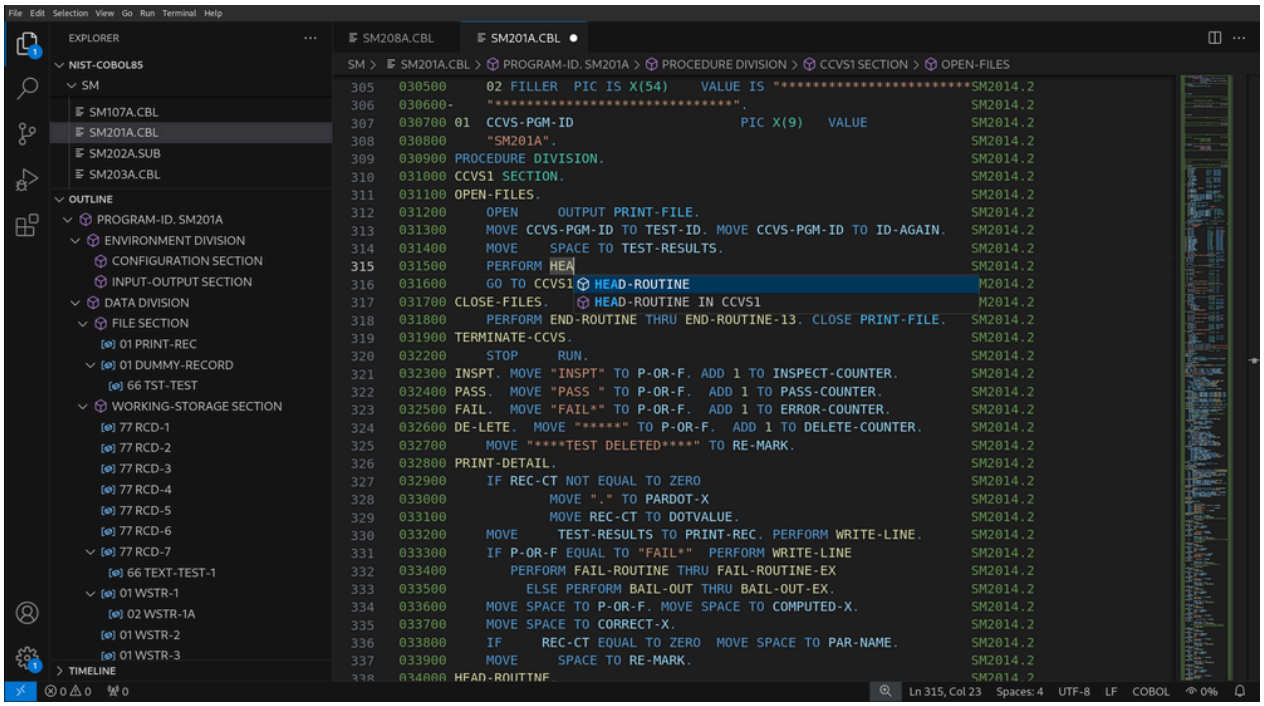


Figure 16: IntelliSense paragraph

(Temporary limitation)

Suggestions of user-defined words may not comprise symbols that are defined in the communication, report, or screen section of the data division. Although user-defined words that occur in configured copybooks are also suggested, preprocessor-related variables or phrases are not.

6.2 Rename Data Items, Sections, and Paragraphs

You can rename any data item by pressing F2 while your cursor is positioned on one of its references. The extension will warn you if a reference to the renamed item appears in a copybook (in which case the renaming of every reference is not performed).

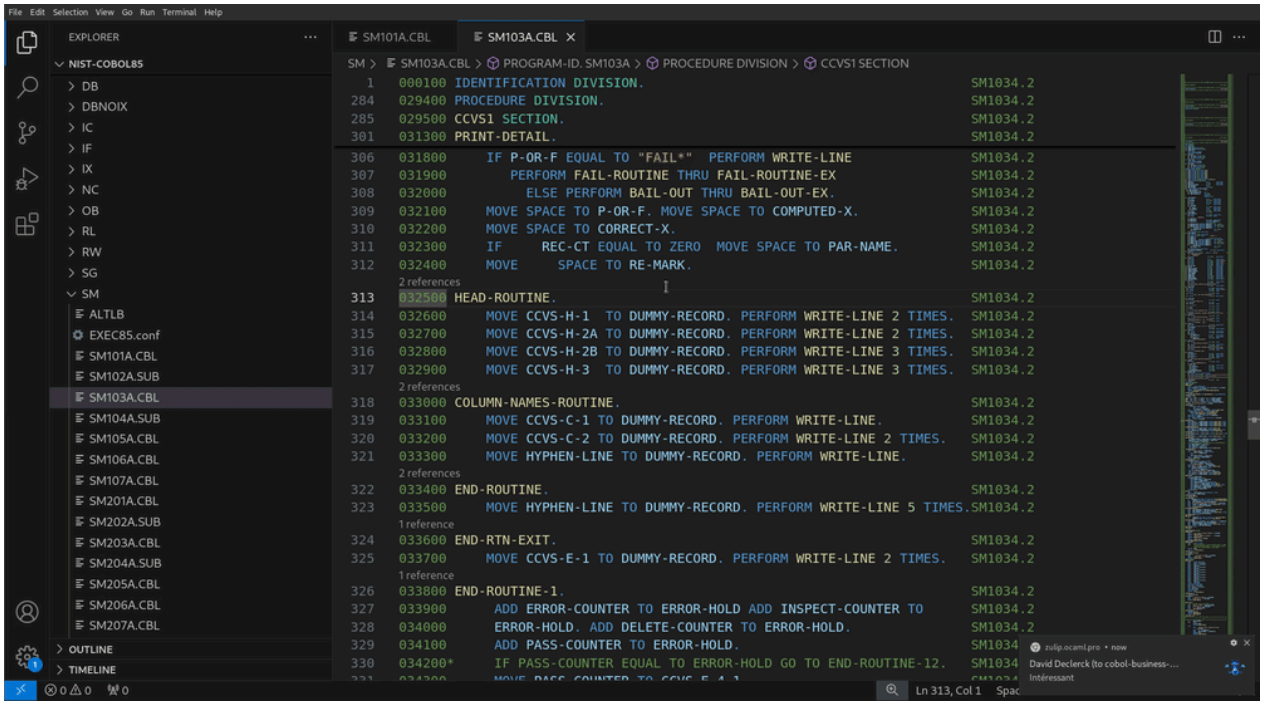


Figure 17: Rename Symbol

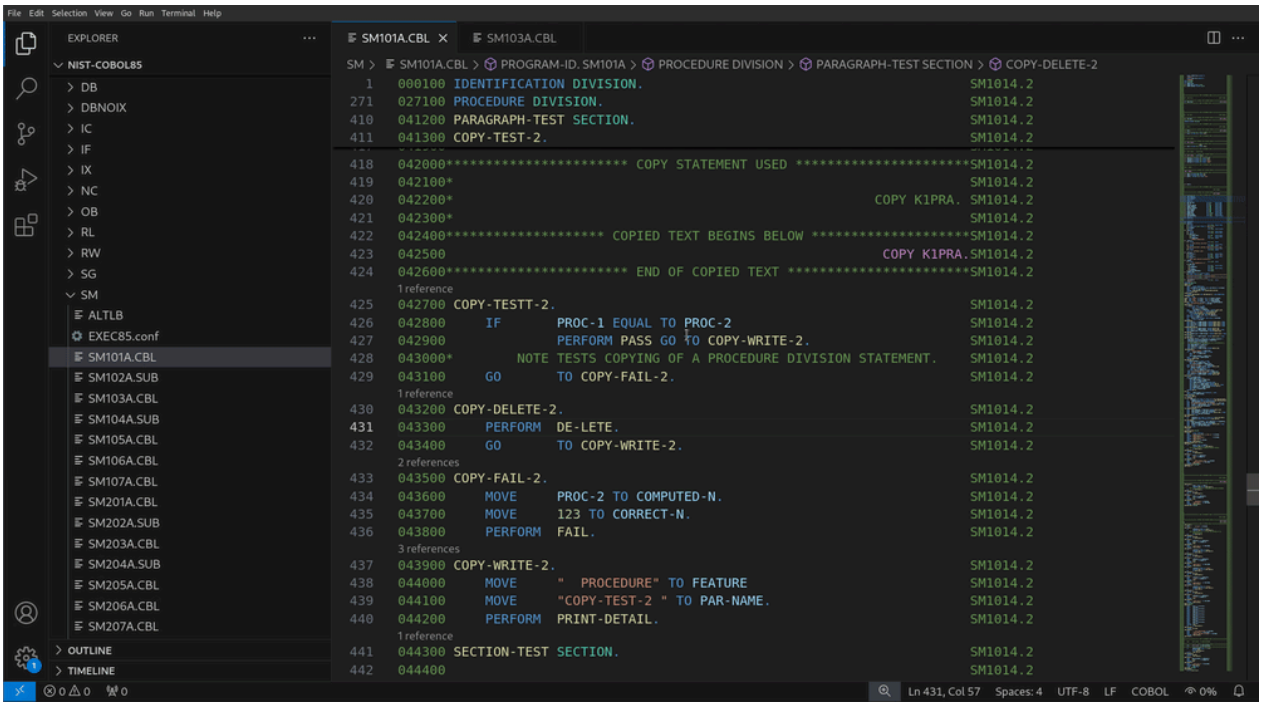


Figure 18: Rename Symbol in Copybook

Sections and paragraphs of the procedure division can also be renamed in the same way.

The same limitations as for Go to References apply to this feature.

7 Control-Flow Visualization

7.1 Exploring the Control-flow

Navigating a graphical representation of a COBOL program's control-flow proves invaluable when it comes to decipher its overall logic. To do this, open the command palette (or type Ctrl+Shift+P), and select `SuperBOL: Show Control-flow` (you can also right click and select `Show Control-flow` in the menu). You are then presented with a list of portions of program to consider (either the entire program, or individiual sections): select one element to see the corresponding CFG.

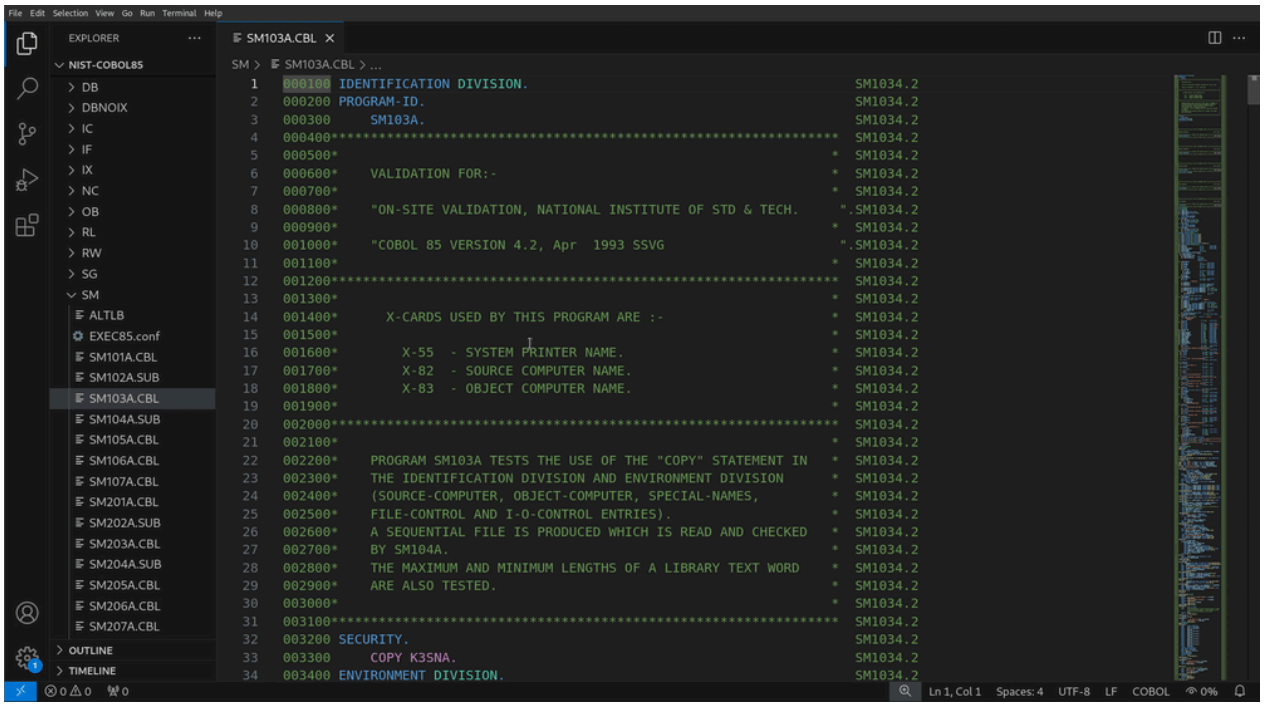


Figure 19: CFG Explorer

Various settings are provided to tune the rendering of CFGs.

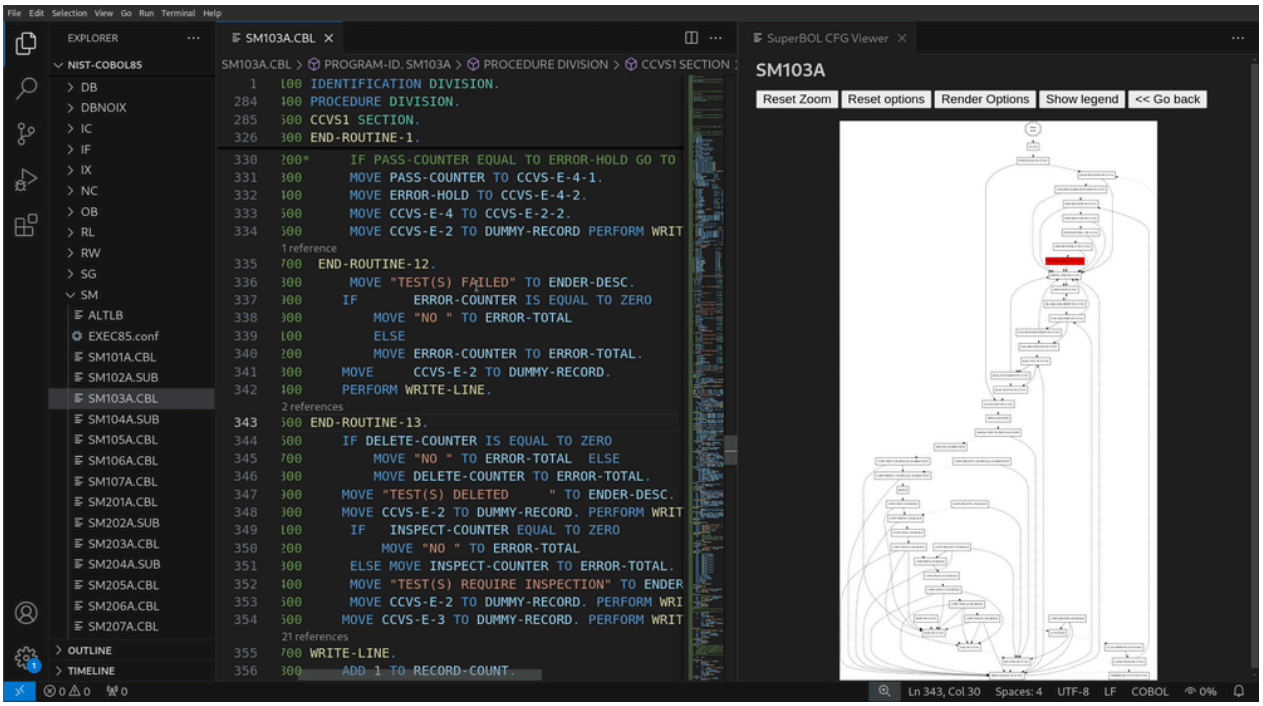


Figure 20: CFG Explorer Collapse Fallthrough

7.2 CFG as an arc diagram

A rendering of CFGs as arc diagrams is also available. In this representation, named sections and paragraphs are laid out vertically, and arcs show the direction of control-flow between them.

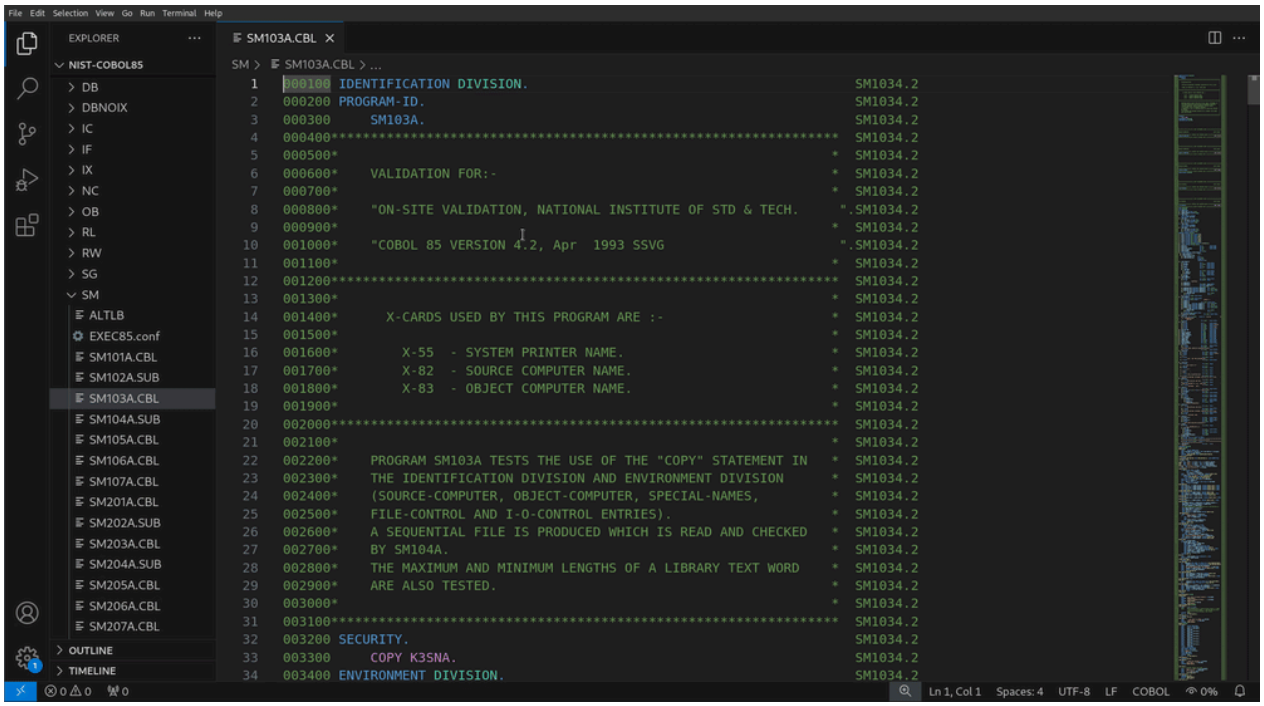


Figure 21: CFG Explorer as Arc Diagram

8 Building Programs

SuperBOL Studio allows you to directly build your COBOL programs with GnuCOBOL.

> **Note**
>
> We recommend that a version of GnuCOBOL that is at least as recent as version 3.2 be available on the system running VSCode. Debug and coverage features respectively assume that gdb and gcov are installed.
>
> On Windows systems, users may employ dedicated installers that are available here. Linux users may rely on their favorite package manager and install `gnucobol`.

All the actions describe below must start with an *opened* COBOL file, so that VSCode knows your configurations are done for the COBOL language, with the SuperBOL extension.

8.1 Simple Build

If you'd like to build your COBOL programs with the default options from `cobc`:

1. press Ctrl+Shift+B, or click on the `Terminal > Run Build Task...` menu option.
2. you will then be presented with several default build options, and select the one you wish to run:

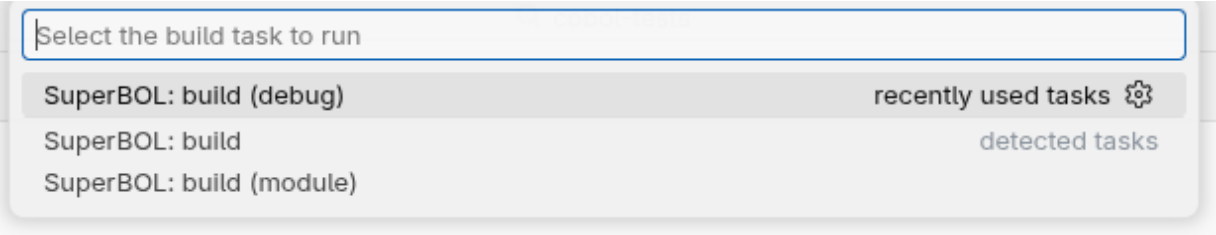


Figure 22: Defaults build options

These options have the following effects, assuming you opened `PROG.cob`:

- `SuperBOL: build`: this option will build your program as an **executable**, **without** debugging informations, it will execute the following `cobc` command:

```
cobc -x -fformat=auto -std=default -ext cpy -ext cbl -ext cob PROG.cob -I .
```

- `SuperBOL: build (debug)`: this option will build your program as **executable**, **with** debugging informations, it will execute the following `cobc` command:

```
cobc -x -ftraceall -g -fformat=auto -std=default -ext cpy -ext cbl -ext cob PROG.cob
-I .
```

- `SuperBOL: build (module)`: this option will build your program as a **module**, **without** debugging informations, it will execute the following `cobc` command:

```
cobc -m -fformat=auto -std=default -ext cpy -ext cbl -ext cob PROG.cob -I .
```

8.2 Settings that can change the execution of build tasks

Some of the **User** or **Workspace** settings have an influence on the commands runned by the build tasks. You can look at the list of available options to learn more about them.

The options that have an effect on the build tasks are:

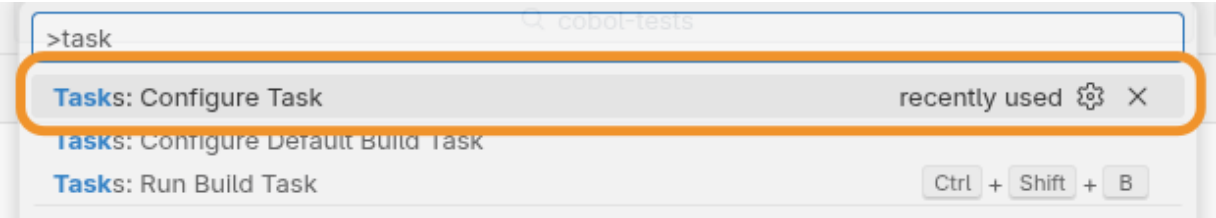
- **superbol.cobol.dialect**: it will set the `-std` option of `cobc`
- **superbol.cobol.sourceFormat**: it will set the `-fsource-format` option of `cobc`.
- **superbol.cobol.copybooks**: it will add `-I` flags to `cobc`, one for each value in the array.
- **superbol.cobcPath**: it will set the `cobc` path that will be executed to build your programs.

8.3 Configuring a build task

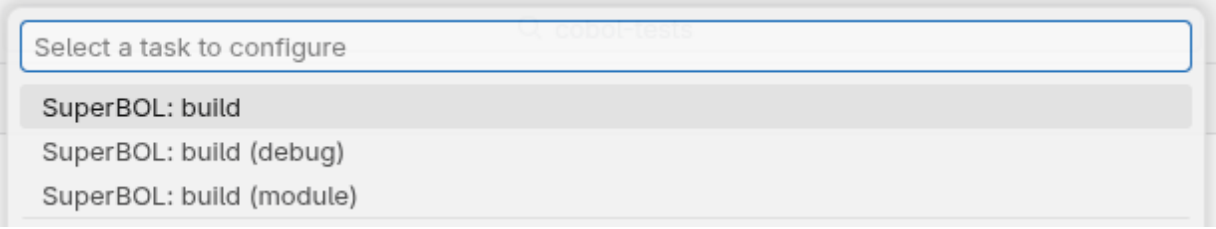
If you wish to perform more complex building tasks, you can create your own building task derived from any SuperBOL provided build tasks.

To do so you can:

1. open the VSCode command palette with Ctrl+Shift+P
2. select the option `Tasks: Configure Task`:



3. you should then be presented with the list of SuperBOL defaults building tasks:



4. from this list you can select any of these task, but to reduce the amount of options to be edited, you can choose the one that fits the best the build actions you'd like to run, if unsure you can select `SuperBOL: build`

VSCode will then automatically create a `.vscode/tasks.json` file, filled with the task you selected. If the file already existed the task you selected will then be added to the list of tasks.

Assuming you selected the `SuperBOL: build (debug)` task earlier you should be presented with the following content:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "superbol",
      "forDebug": true,
      "forCoverage": false,
      "executable": true,
      "extraArgs": [],
      "problemMatcher": [
        "$gnucobol",
        "$gnucobol-warning",
        "$gnucobol-error",
        "$gnucobol-note"
      ],
      "group": "build",
      "label": "SuperBOL: build (debug)"
    }
  ]
}
```

The different fields available in this build configuration are also available in the other provided build configurations, but with different values. Here is a description of the meaning of all fields in this configuration:

- `"type"`: This field informs VSCode that this configuration is to be used with the SuperBOL extension
- `"forDebug"`: if set to `true`, then the program will be build with debugging informations, otherwise the program will be build as a release program.
- `"forCoverage"`: if set to `true`, then the program will be build with coverage informations, otherwise the program will be build as a normal program.

- `"executable"`: if set to `true`, then the program will be build as an executable binary, otherwise the program will be build as a module to be executed with a COBOL runner (usually `cobcrun` for GnuCOBOL modules.)
- `"extraArgs"`: you can add strings to this array that will be passed to the GnuCOBOL compiler to be used when building your program.
- `"problemMatcher"`: this field is not to be edited, it enables VSCode to highlight the sources of a build error.
- `"group"`: this field should also not be edited, it informs VSCode that this task is used to build a program.
- `"label"`: this field contain the name of the build task, it might be used in other VSCode files configurations, so you can set it to the name you see fit the best.
- `"cobcPath"`: this field is **hidden** by default, but you can set the path to your `cobc` compiler if it is not directly available in your *PATH*.

With all these fields you should be able to configure your build task to best fit your needs.

8.4 Building with a custom configuration

Once you have configured your build task, you can use it to build your COBOL programs.

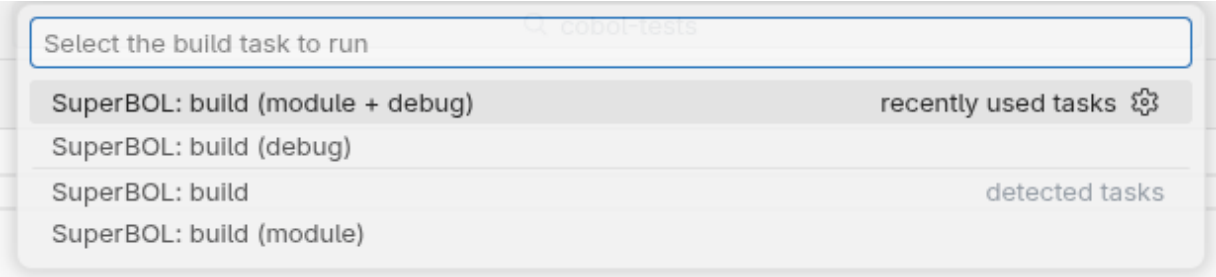
As an example let's suppose that you have the following task defined:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "superbol",
      "forDebug": true,
      "forCoverage": false,
      "executable": false,
      "extraArgs": [],
      "problemMatcher": [
        "$gnucobol",
        "$gnucobol-warning",
        "$gnucobol-error",
        "$gnucobol-note"
      ],
      "group": "build",
      "label": "SuperBOL: build (module + debug)"
    }
```

```
    ]
}
```

To use your customized build task task:

1. open the COBOL program you wish to build
2. press Ctrl+Shift+B
3. select the label of your custom build task (`SuperBOL: build (module + debug)` in our example):



8.5 Selecting a default build task

If you have a task you wish to be executed as the default build task:

1. press Ctrl+Shift+P
2. select `Tasks: Configure Default Build Task`
3. select the task you wish to be executed as the default build task
4. open your COBOL program sources
5. press Ctrl+Shift+B
6. VSCode will execute automatically the task you selected as default

8.6 Debugging your programs

Once your programs are build, you may wish to check Debugging Programs to see how you can debug your COBOL programs.

8.7 Examples of custom configurations

Here are a few examples of configurations you might want to set up. In your `.vscode/tasks.json` you should add them to the `"tasks"` array.

8.7.1 Building GnuCOBOL module with debugging informations

```
{
  "type": "superbol",
  "forDebug": true, // generate debug informations
  "forCoverage": false,
  "executable": false, // build a module
  "extraArgs": [],
  "problemMatcher": [
    "$gnucobol",
    "$gnucobol-warning",
    "$gnucobol-error",
    "$gnucobol-note"
  ],
  "group": "build",
  "label": "SuperBOL: build (module + debug)"
}
```

8.7.2 Building GnuCOBOL executable to check coverage

```
{
  "type": "superbol",
  "forDebug": false,
  "forCoverage": true, // generate coverage informations
  "executable": true, // build an executable
  "extraArgs": [],
  "problemMatcher": [
    "$gnucobol",
    "$gnucobol-warning",
    "$gnucobol-error",
    "$gnucobol-note"
  ],
  "group": "build",
  "label": "SuperBOL: build (coverage)"
}
```

8.7.3 Sending extra arguments to `cobc`

```
{
  "type": "superbol",
  "forDebug": false,
  "forCoverage": false,
  "executable": true,
```

```
  "extraArgs": [
    "-fec=all" // pass `-fec=all` to `cobc`
  ],
  "problemMatcher": [
    "$gnucobol",
    "$gnucobol-warning",
    "$gnucobol-error",
    "$gnucobol-note"
  ],
  "group": "build",
  "label": "SuperBOL: build (all exceptions)"
}
```

9 Debugging Programs

9.1 Debugging

In order to debug a COBOL program, you first need to run a *build task* with appropriate debug options. Once this is done, you can *launch* the compiled program in a debugging session.

You can learn more about SuperBOL build tasks in the Building Programs chapter.

> **i** Note
>
> We recommend that a version of GnuCOBOL that is at least as recent as version 3.2 be available on the system running VS Code. Debug and coverage features respectively assume that gdb and gcov are installed.
>
> On Windows systems, users may employ dedicated installers that are available here. Linux users may rely on their favorite package manager and install `gnucobol`.

9.2 Launching the Compiled Program for Debugging

If needed, you can place a breakpoint on statements (or paragraph titles in the procedure division) by clicking on the red dot that appears when you hover the cursor on the left margin (or with F9). Click on the red dot or press F9 again to remove a breakpoint. Then, to launch the program in debug mode, select `Run > Start Debugging` (F5). This will run your program until a first breakpoint if reached, or to completion.

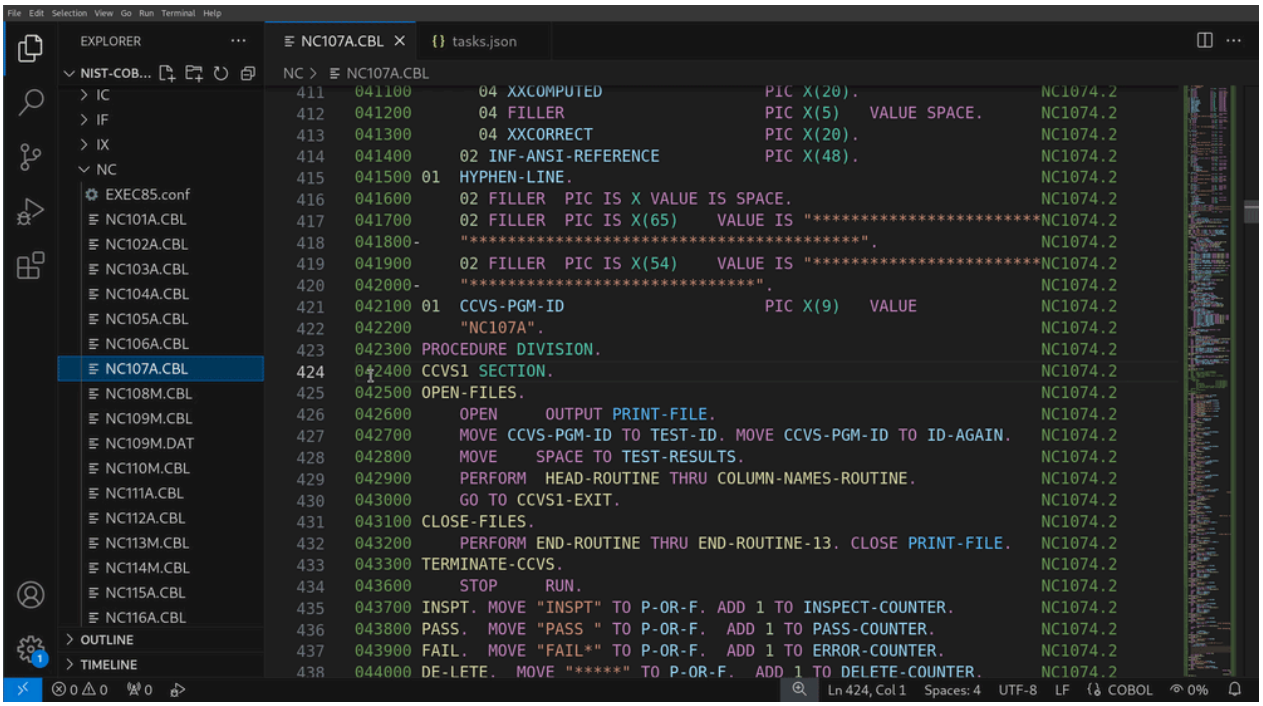


Figure 23: Start Debugging

Once stopped on a breakpoint, you can investigate the values of data items from the program using the `VARIABLES` panel on the left-hand side.

Press F10 to step to the next statement, or F5 again to continue until the next breakpoint, or termination of the program.

9.3 SuperBOL debug configurations

SuperBOL can handle several debugging scenarios, such as launching a program to debug it, attaching to a running process or even attaching to a remote process to debug it.

In order to select which scenario SuperBOL should run, you have to provide a list of configurations stored in the `.vscode/launch.json` file. This file can be generated with the following steps:

1. Open the `Run and Debug` panel (Ctrl+Shift+D)

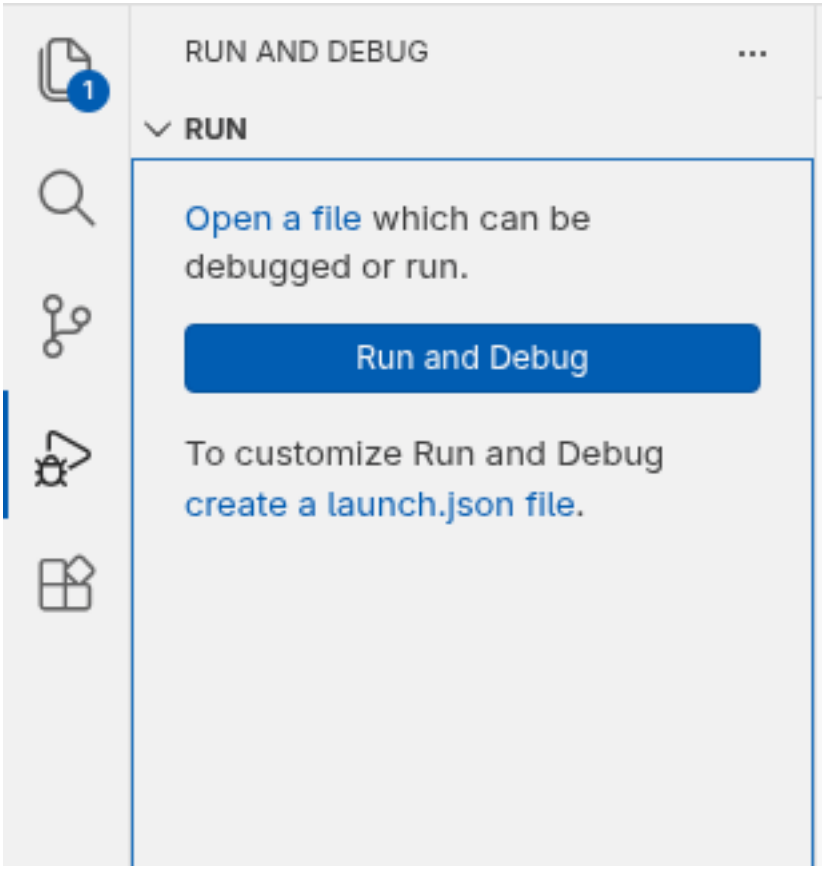
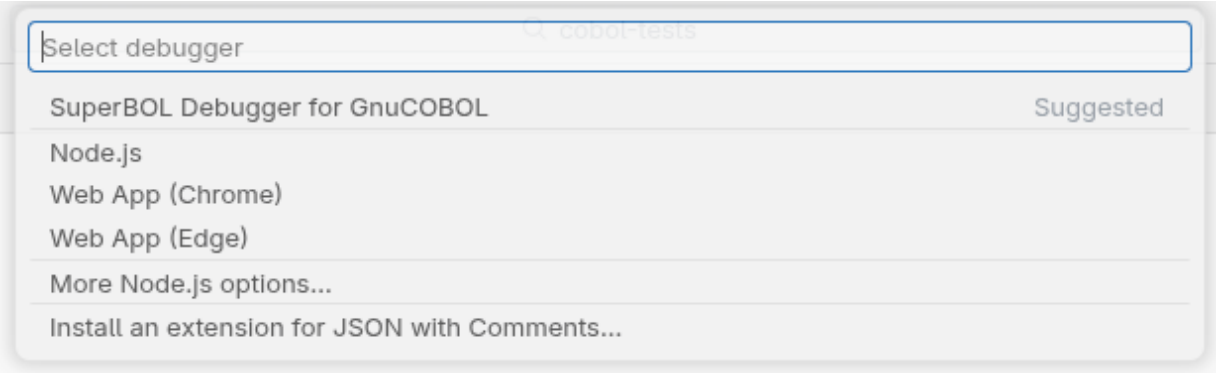


Figure 24: Run and Debug

2. Click the `create a launch.json file` link. If you don't have this view, this means that you already have such a file, you should be able to open it now, and skip this setup.
3. A list of presets will be presented, you should select the `SuperBOL Debugger for GnuCOBOL`.



4. The `.vscode/launch.json` will be generated with preset configurations provided by superbol:

```
{
   // Use IntelliSense to learn about possible attributes.
   // Hover to view descriptions of existing attributes.
   // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
   "version": "0.2.0",
   "configurations": [
       {
           "name": "SuperBOL: debug (launch)",
           "type": "superbol-gdb",
           "request": "launch",
           "preLaunchTask": "SuperBOL: build (debug)",
           "target": "${file}",
           "arguments": ""
       },
       {
           "name": "SuperBOL: debug (attach local)",
           "type": "superbol-gdb",
           "request": "attach",
           "pid": "${input:pid}",
           "target": "${file}"
       },
       {
           "name": "SuperBOL: debug (attach remote)",
           "type": "superbol-gdb",
           "request": "attach",
           "remoteDebugger": "${input:remoteDebugger}",
           "target": "${file}"
       }
   ]
}
```

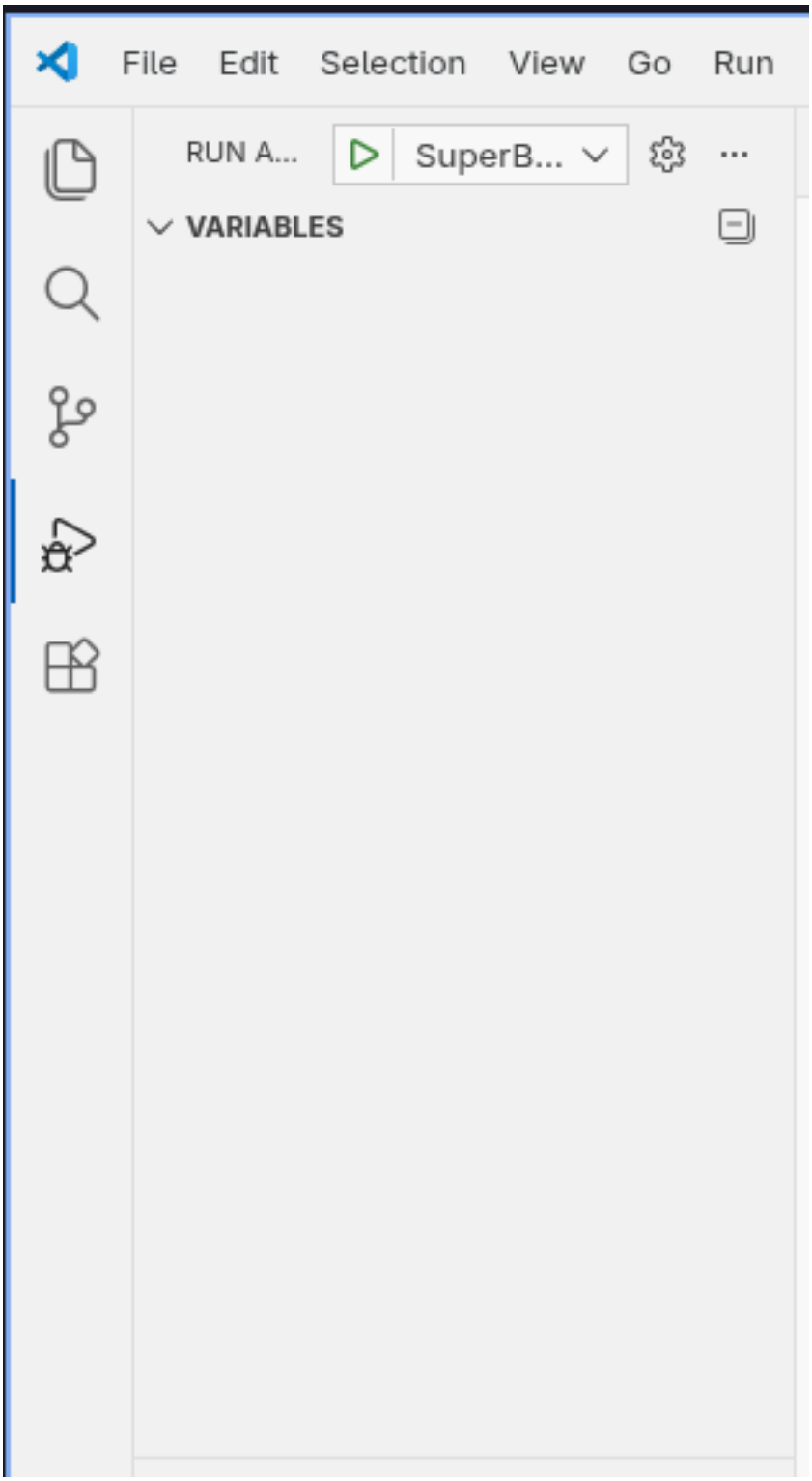
With those presets, you can handle the different scenarios presented earlier:

- `"SuperBOL: debug (launch)"` to compile the currently opened program, and to run `gdb` on the compiled program.
- `"SuperBOL: debug (attach local)"` will ask you for a `pid` to attach the `gdb` debugger to, and use the currently opened COBOL file as the source code of the attached process.
- `"SuperBOL: debug (attach remote)"` will ask you for an adress and port to connect to a remote debugger, with `gdbserver`, and use the currently opened COBOL file as the source code of the remote process.

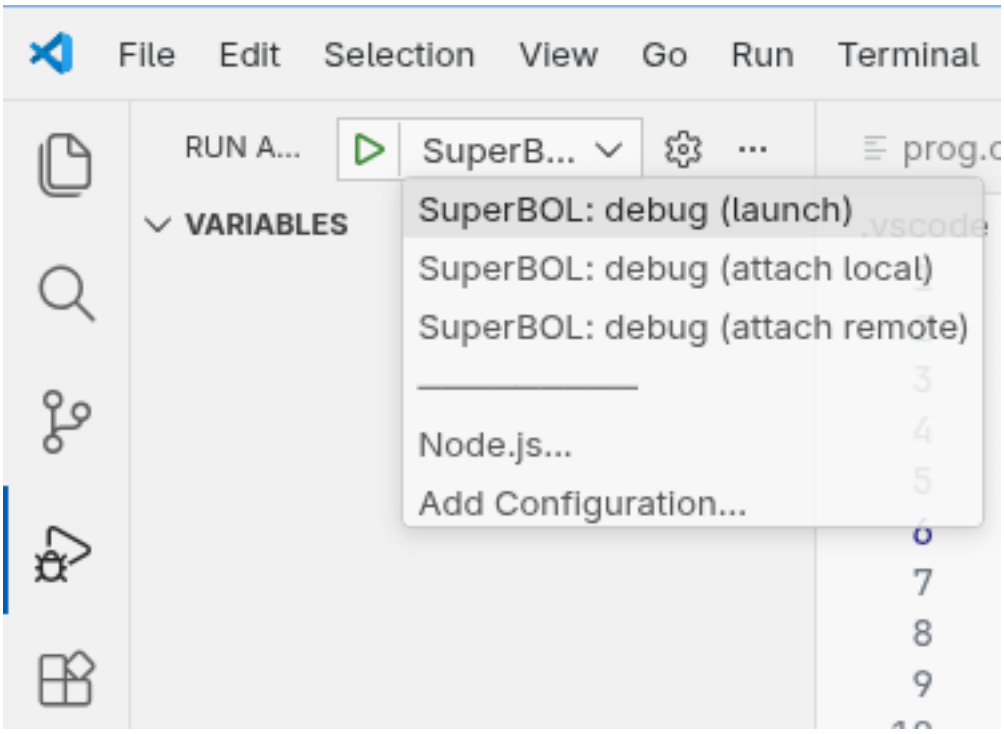
> **Note**
>
> In all the scenarios presented, you must have the COBOL source on your local system, and have the program you wish to debug opened in your editor.
>
> For the `"SuperBOL: debug (attach remote)"` configuration, the remote server should have `gdbserver` installed and available.

To select which configuration to use to debug your program you should:

1. Open the `Run and Debug` panel (Ctrl+Shift+D)



2. Select the configuration in the dropdown menu at the top of the panel



To start your debugging you can then press the F5 key. The configuration you selected should be run by default.

Both `remote` configurations needs input to successfully launch, this requires some manual configuration, which is explained in the next section.

9.4 Managing configuration inputs

As you may have noticed, some fields in the SuperBOL configurations have an `${input:<identifier>}` parameter. This means that you must provide an input when the configuration is launched.

VSCode does not provide a way to automatically configure those inputs so you must write some configurations yourself. In order to setup those inputs, you should:

1. Open the `.vscode/launch.json` file
2. Add an `inputs` array in the file:

```
{
    "version": "2.0.0",
    "configurations": [
        ...
    ],
```

```
    "inputs": []
}
```

3. In this array, add every input needed by the debug configurations, in our case they are `"pid"` and `"remoteDebugger"` (you can only add one of both if you don't intend to use both of them.) Your inputs array should look like so:

```
{
   "inputs": [
       {
           "id": "pid",
           "type": "promptString",
           "description": "PID of process to attach to."
       },
       {
           "id": "remoteDebugger",
           "type": "promptString",
           "description": "Address of the remote debugger."
       }
   ]
}
```

   In this array, each objects have the same fields, with different values:

   - `"id"`: The identifier of the input, must be the same as the identifier on the right of the `${input:<identifier>}` in your configuration.
   - `"type"`: The type of input to provide, here `"promptString"` means you will be asked to input a string before the configuration can be launched.
   - `"description"`: A small description of the input to provide.

Now when you use the `"attach"` configuration VSCode will ask you for the corresponding input.

9.5 Customizing the debug configurations

The presets configurations might not be suited to your use case, but you can customize them to fit your needs.

To customize your debug configuration, you can either edit the presets or add a new one in the `.vscode/launch.json` file. Here are the fields you can add or edit in the configurations:

- `"name"` (*mandatory*): The name of the configuration, you can set it to the name you wish.

- `"type"` (*mandatory*): This is the only constant field, that should always be set to `"superbol-gdb"`.
- `"request"` (*mandatory*): This should be either:
  - `"launch"`: this means that the debugger will run over a target that it launches
  - `"attach"`: this means that the debugger will attach to an already running process
- `"preLaunchTask"`: This field indicates which task should be run before a `"launch"` request. This should be set to the label of the build task that should be run before launching the debugger. You can check Configuring a build task for more informations on those task.
- `"target"`: Path to the program to be debugged. You can set it to `"${file}"` to debug the program that is opened in your editor
- `"arguments"`: An array of arguments that should be passed to the debugged program. In an `"attach"` request, these arguments will be passed only when the programs restarts.
- `"cwd"`: The directory from which the debugger should be started
- `"env"`: An object containing environment variables, for example:

```
{
    "COB_LIBRARY_PATH": "/usr/lib/cobol/modules"
}
```

- `"useCobcrun"`: Use a cobol module loader, you can specify the loader to use in `"cobcrunPath"`.
- `"cobcrunPath"`: Path to the cobol module loader, `"cobcrun"` by default
- `"sourceDirs"`: An array of path to specify where to look for the source code. When this array is not provided, only the current directory is concidered.

Some options are specific to a type of `"request"`:

- in `"launch"` requests you can set:
  - `"gdbtty"`: This field enable you to use a different terminal for the COBOL input and ouput. It can be set to `false` to use the integrated debug terminal, `true` to use an external terminal, which is choose from `xterm`, `gnome-terminal`, `xfce4-terminal`, or `konsole`. You can also set the field to one of the following string:
    - `"vscode"`: Use VSCode terminal.
    - `"xterm"`, `"gnome-terminal"`, `xfce4-terminal"`, `"konsole"`: Use the terminal specified by the string.
    - `"external"`: Use one of previously listed terminals, the first one that is available to be executed will be used.
- in `"attach"` requests you can set:

- `"pid"`: The PID of the program to be attached. You should use an `${input:<identifier>}` variable if the PID of the program is not constant.
- `"remoteDebugger"`: Address of `gdbserver`, in a `"host:port"` format.

9.6 Example of debug configurations

These different configurations are in the `"configurations"` array of your `.vscode/launch.json` file.

9.6.1 Debug a GnuCOBOL module, with a custom `cobcrun` path

```
{
    // Custom name for the configuration
    "name": "SuperBOL: debug (module)",

    "type": "superbol-gdb",

    // The debugger will launch `cobcrun`
    "request": "launch",

    // We use a task to build a GnuCOBOL module with debug info
    "preLaunchTask": "SuperBOL: build (module + debug)",
    "target": "${file}",

    // Use a GnuCOBOL module loader
    "useCobcrun": true,

    // Use a specific cobcrun, given by its path
    "cobcrunPath": "/opt/gnucobol/bin/cobcrun"
}
```

10 Code Coverage

GnuCOBOL can instrument your programs so they can generate coverage information at runtime. To enable this feature, you can set the `forCoverage` setting to `true` in the `Superbol: build (debug)` task in your `tasks.json` file (see Building programs for more informations on tasks.) This flag instructs the extension to pass the `--coverage` flag to the `cobc` compiler.

10.1 Coverage Information

Generated coverage files are in `gcov` format; they are portable, and you can use them as you would any other coverage file generated for programs written using other programming languages.

10.2 Highlighting Coverage Information

Coverage data can be shown after the execution of a program that was compiled to generate this information terminates. SuperBOL will display coverage on a line-by-line basis, by highlighting the lines of your source code using colors that represent their coverage status. To enable coverage highlighting, you can open the command palette (or type Ctrl+Shift+P), and select `SuperBOL: Show Coverage`. You can also hide the highlighting with the command `SuperBOL: Hide Coverage`, and update it after re-executing your program with `SuperBOL: Update Coverage`.

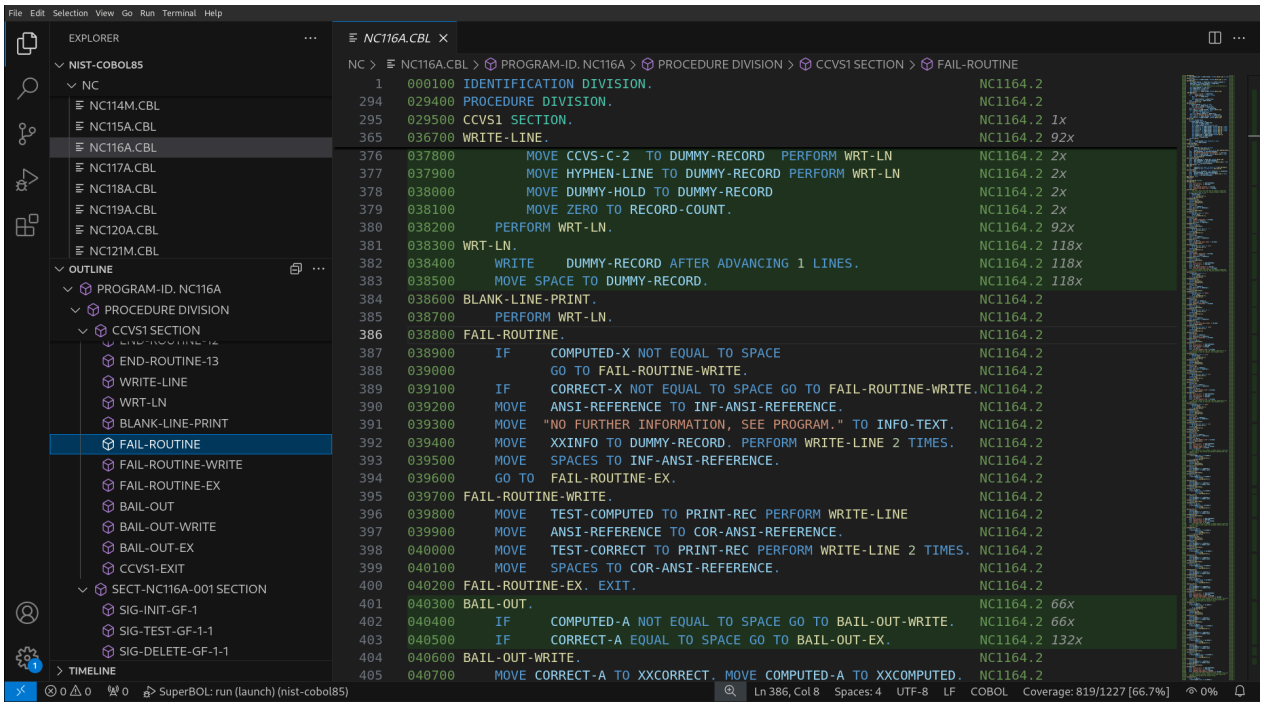


Figure 25: Show Coverage

11 Using Vscode in General

11.1 Navigating through Warnings and Errors aka Problems

Problems (errors and warnings) are not always as visible as we would want. You should probably install an extension to improve their display, typically the “Error Lens” VSCode extension that displays errors as inline messages in the source code.

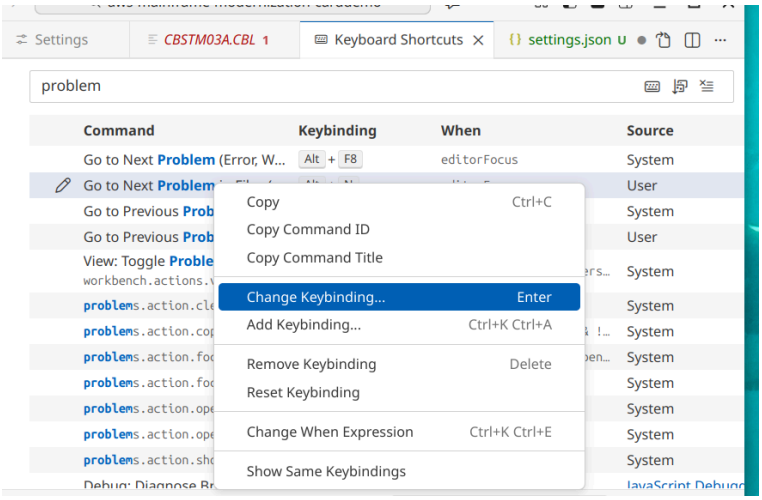
You can also use the following shortcuts:

- Ctrl+Shift+M: go to the first problem, and display the “Problems” tab, showing the list of all problems found. You can use arrow keys to navigate the problems.
- F8: go to next problem in file
- Shift+F8: go to previous problem in file

Some users like to use other shortcuts, for example because F8 is already used for something else.

For that:

- Open the “Keyboard Shortcuts” by typing successively Ctrl +K Ctrl+S
- Type “Problem” to display related shortcuts
- Click on the shortcut you want to edit, and use “Change Keybinding” to propose a new binding
- For example, in our case, we will use Alt+N for “Go to Next Problem in Files”



11.2 Switching between Light and Dark Themes

1. Open the **Command Palette** with CTRL + SHIFT + P
2. Type “Color Theme”.
3. Select **Preferences: Color Theme** from the dropdown.
4. Choose a color theme, typically the **Dark modern** or **Light modern**.

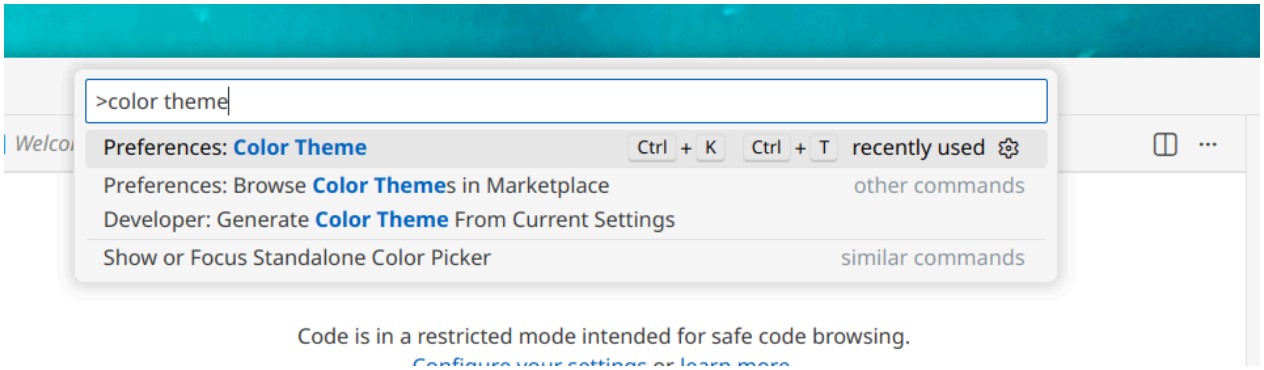


Figure 26: Color Theme Dropdown

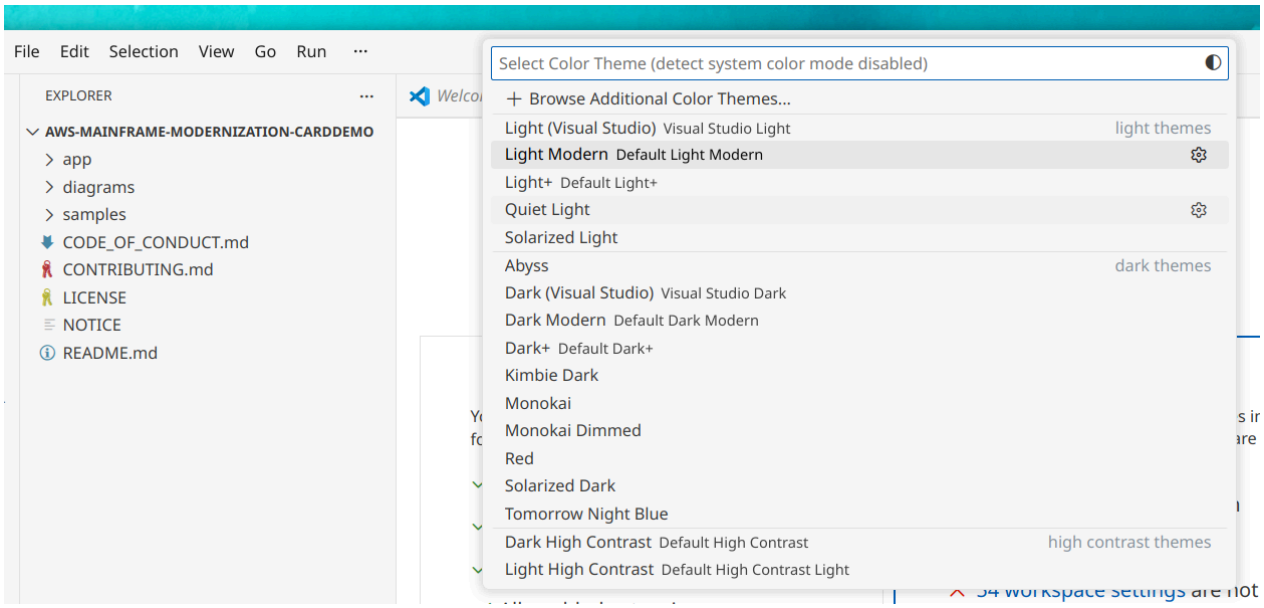


Figure 27: Color Theme Dropdown

11.3 The “Do you trust the authors” popup

When entering a new project, you always get the following popup:

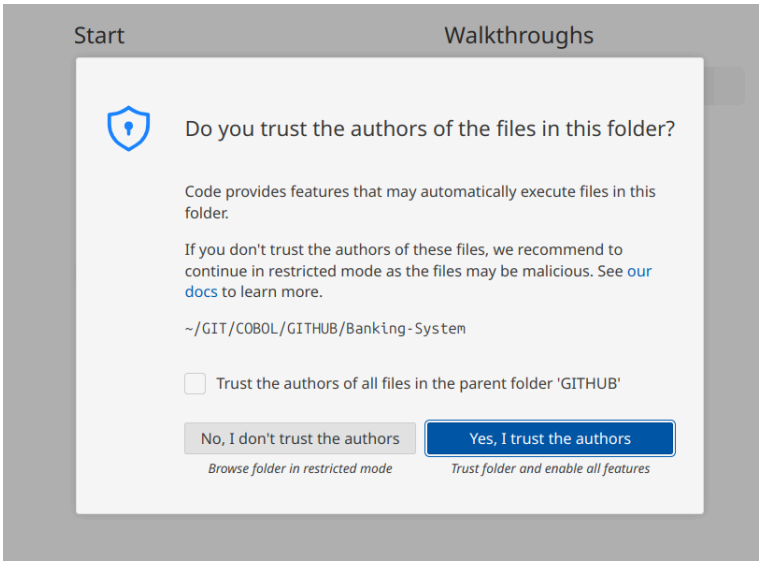


Figure 28: Do you trust popup

To use SuperBOL Studio extension, you need to reply “Yes, I trust the authors”.

If you reply “No, I don’t trust the authors”, Vscode enters a **Restricted Mode**, where:

- Extensions are disabled: Most extensions that actually do things (like debuggers, linters, or language servers for Python/C++) will be turned off. This prevents a malicious workspace from using an extension to run a script in the background.
- Tasks are blocked: Any automation you’ve set up in .vscode/tasks.json (like your make command for Pandoc) will be disabled. VS Code won’t run them because it doesn’t know if the command is safe.
- Debugging is disabled: You cannot launch the debugger, as this requires executing the code in the workspace.
- Workspace Settings are ignored: Some settings defined specifically for that folder (in .vscode/settings.json) will be overridden by your global user settings to prevent the folder from “hijacking” your editor’s behavior.

If you replied “No”, you will see a “Restricted Mode” button in the bottom left corner, click on it and select the “Trust” box to change it.

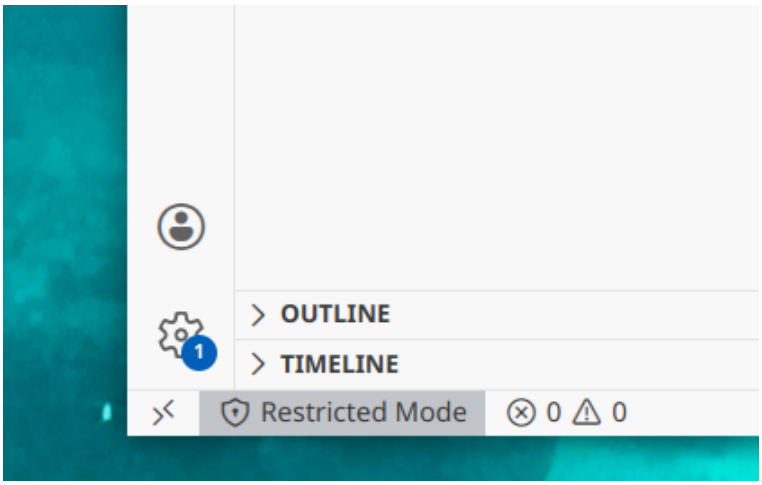


Figure 29: Restricted Mode

12 Adding a SuperBOL License

12.1 SuperBOL Premium Features

Some SuperBOL features are accessible only to users with a premium license.

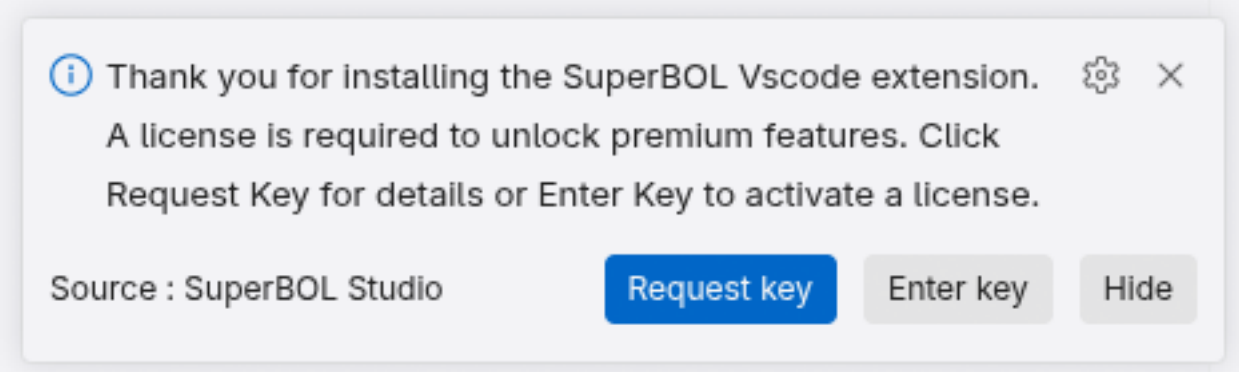
You can purchase one of these licenses by contacting us on our website superbol.eu or by email at superbol@ocamlpro.com.

12.2 Registering a license in VS Code

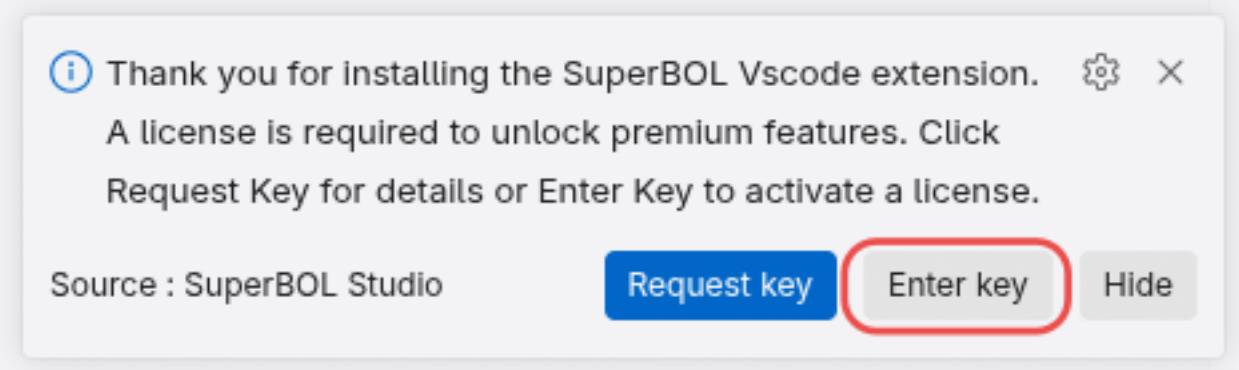
To register a license in your VS Code application, you can either go through the SuperBOL extension `settings` or click on the notification that appears when SuperBOL starts.

12.2.1 Register via the Startup Notification

When starting SuperBOL, if no license is registered, this notification appears in the bottom right corner of the VS Code window:

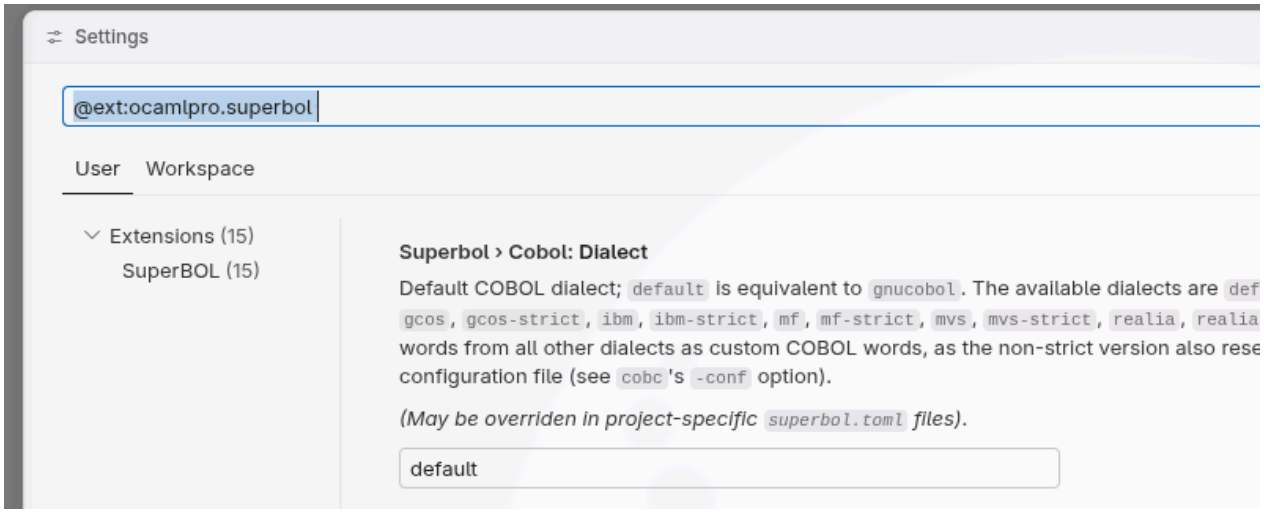


By clicking on `Enter key`, the settings window opens. You can then proceed by following the steps in the adding a license section.



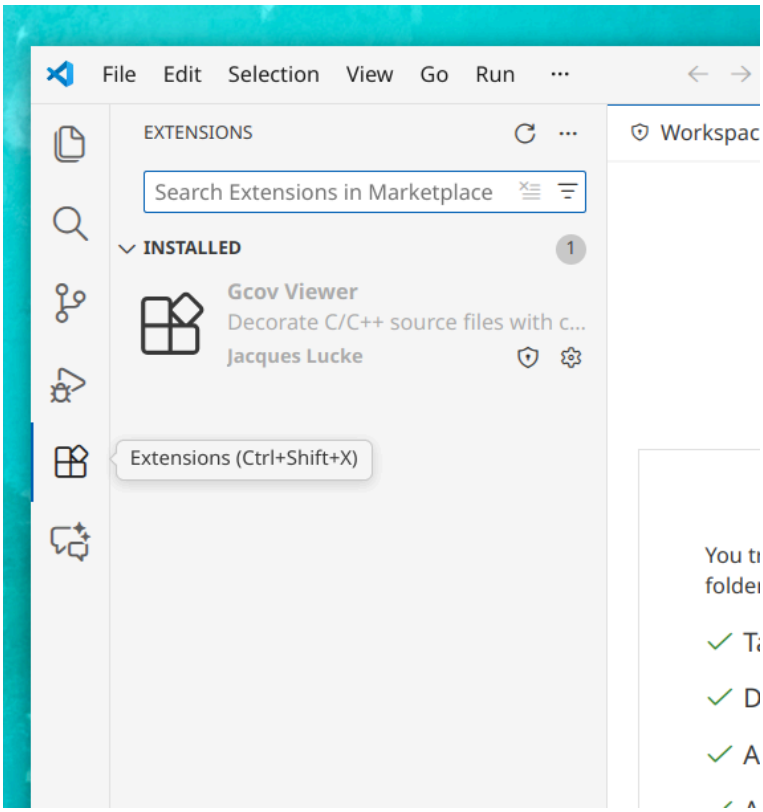
12.2.2 Register via the SuperBOL Extension Settings

You can access the settings with the keyboard shortcut Ctrl+,, and then you can search `@ext:ocamlpro.superbol` to display only SuperBOL settings. You can then keep going from step **5** below.

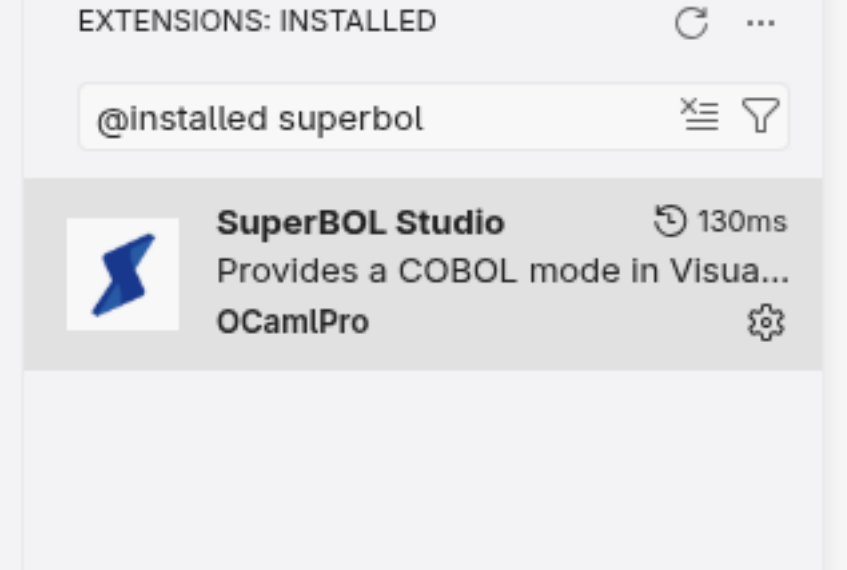


To access the license registration directly through the extension settings:

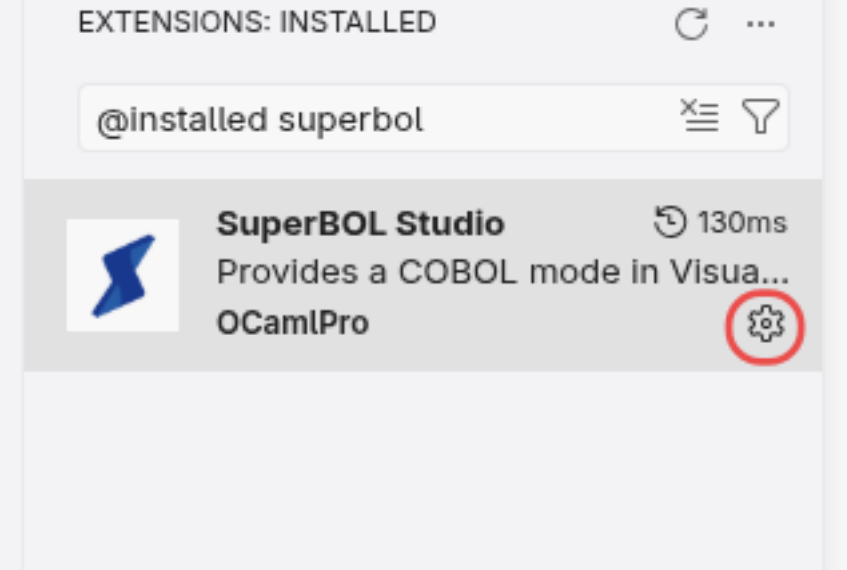
1. Open the Extensions panel:



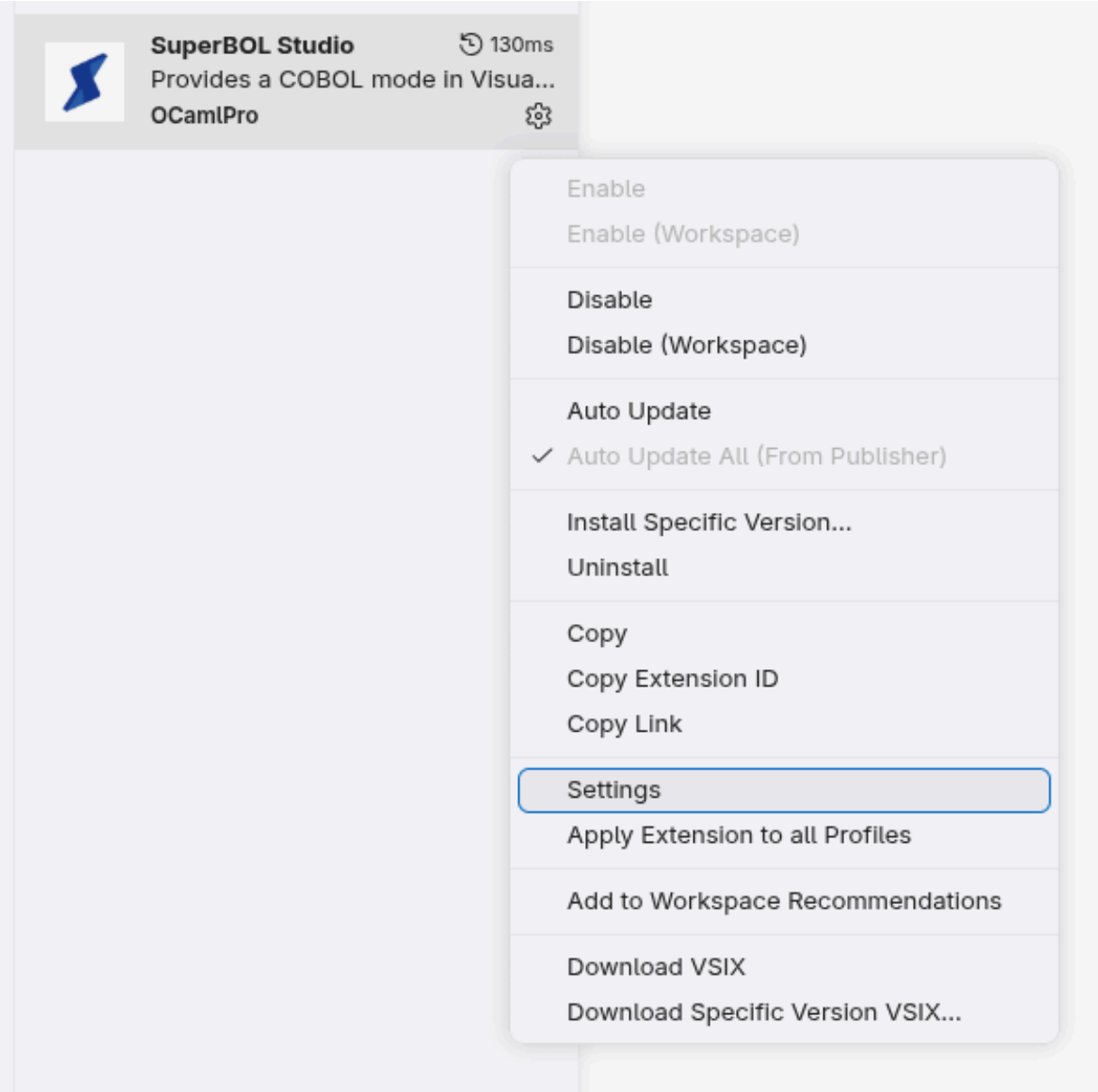
2. Search for “SuperBOL”:



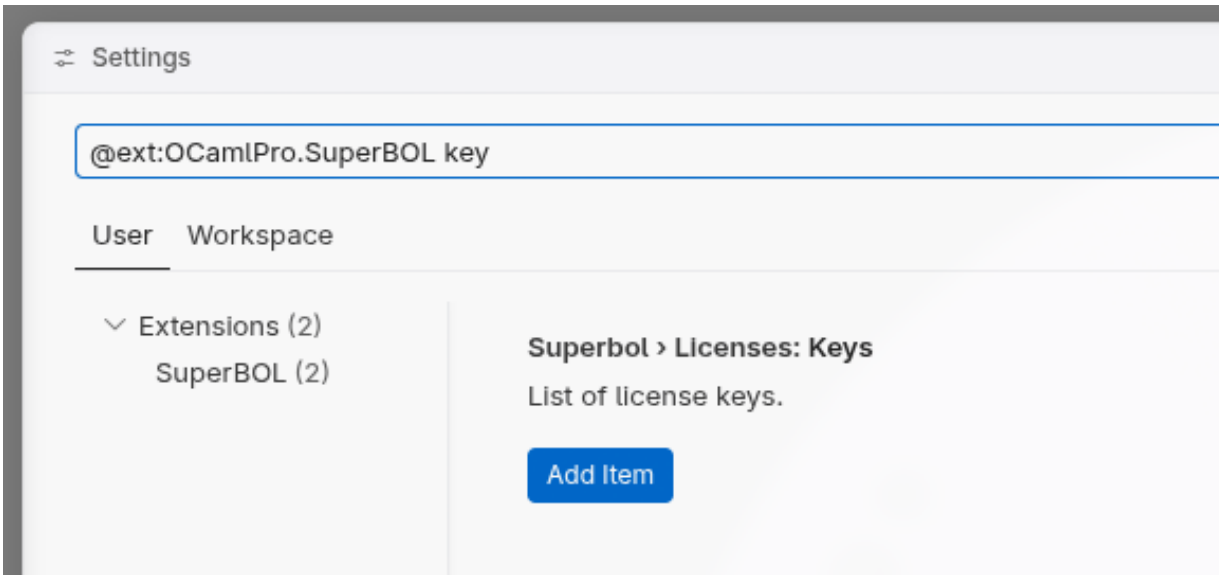
3. Click on the gear icon:



4. Select the “settings” option:

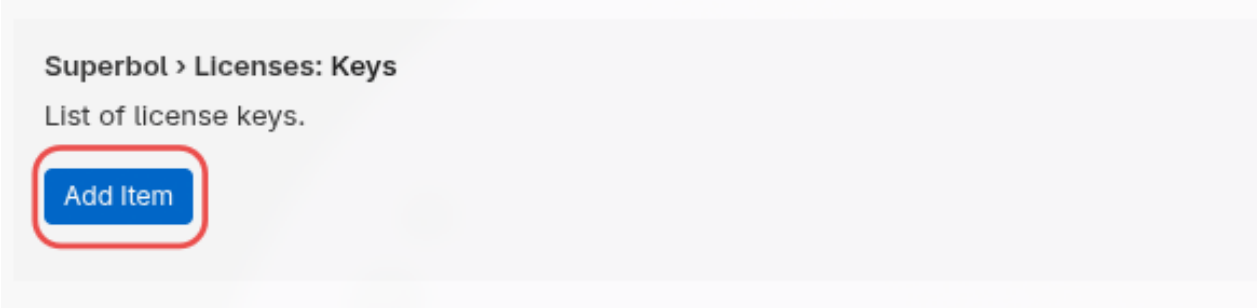


5. Search for “key” in the settings:

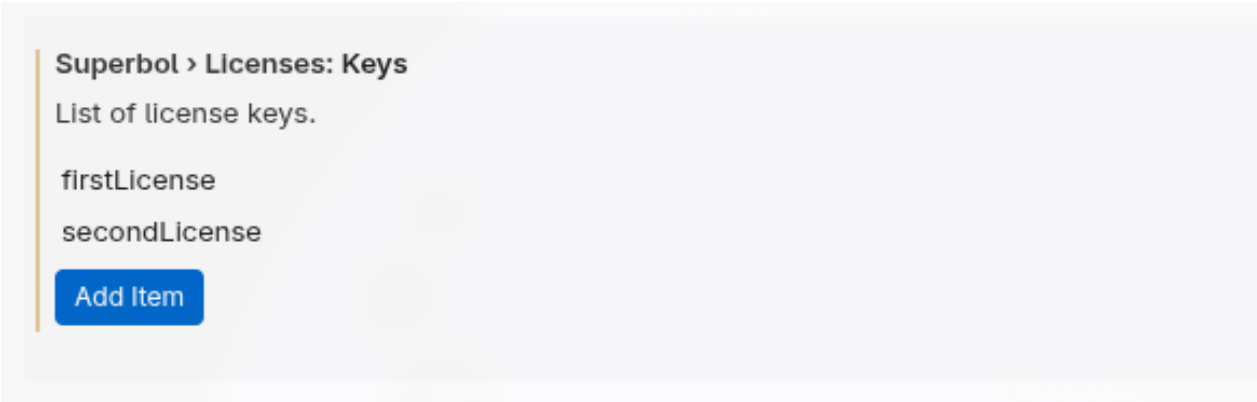


12.2.3 Registering the new SuperBOL license

Once you have access to the licenses settings, you can click on the `Add item` button, a text area will be displayed, in which you can copy your license key.



Only *one* license can be entered at once in a text area. If you own multiple licenses, you must add each key separatly with the help of the `Add item` button.



13 SuperBOL File Viewer Premium

SuperBOL File Viewer is a tool for viewing COBOL data files. It supports the following organization types:

- `SEQUENTIAL` and `LINE SEQUENTIAL`;
- `INDEXED`;
- `RELATIVE`.

These files typically have extensions like `.dat`, `.seq` or `.idx`.

> ! Important
>
> SuperBOL File Viewer is a **Premium** feature. A free version allows you to view files containing at most 100 records. Beyond that, a license key is required (see Adding a SuperBOL License). Contact superbol@ocamlpro.com for commercial offers.

13.1 Prerequisites

COBOL data files cannot be interpreted on their own: their structure is defined by the COBOL program that uses them. To open a data file, you therefore need:

- a **COBOL program** (`.cob` or `.cbl` file) containing the file declaration in the `FILE-CONTROL` section as well as the record descriptions in the `DATA DIVISION`;
- optionally, **copybooks** referenced by this program to describe the field structure. Copybooks are recognized from the extensions listed in the `cobol.copyexts` setting (for example `.cpy`).

You only need to first identify either the COBOL program that defines the data file you wish to view, or the copybook that describes its records: in the latter case, the matching program is searched for automatically in the workspace.

13.2 Opening a File

13.2.1 Step 1: Open the Data File

There are two methods to open a data file in VSCode:

- **Via the file menu**: go to `File > Open File...` and select your data file directly.
- **Via the file explorer**: open the folder containing your files with `File > Open Folder...`, then double-click the desired data file (for example, a file with a `.dat` extension) in the explorer panel on the left.

Once the file is open, a schema selection window appears in the center of the screen.

13.2.2 Step 2: Select a COBOL Program or a Copybook

Opening the file requires associating a COBOL program, or a copybook, that describes its format.

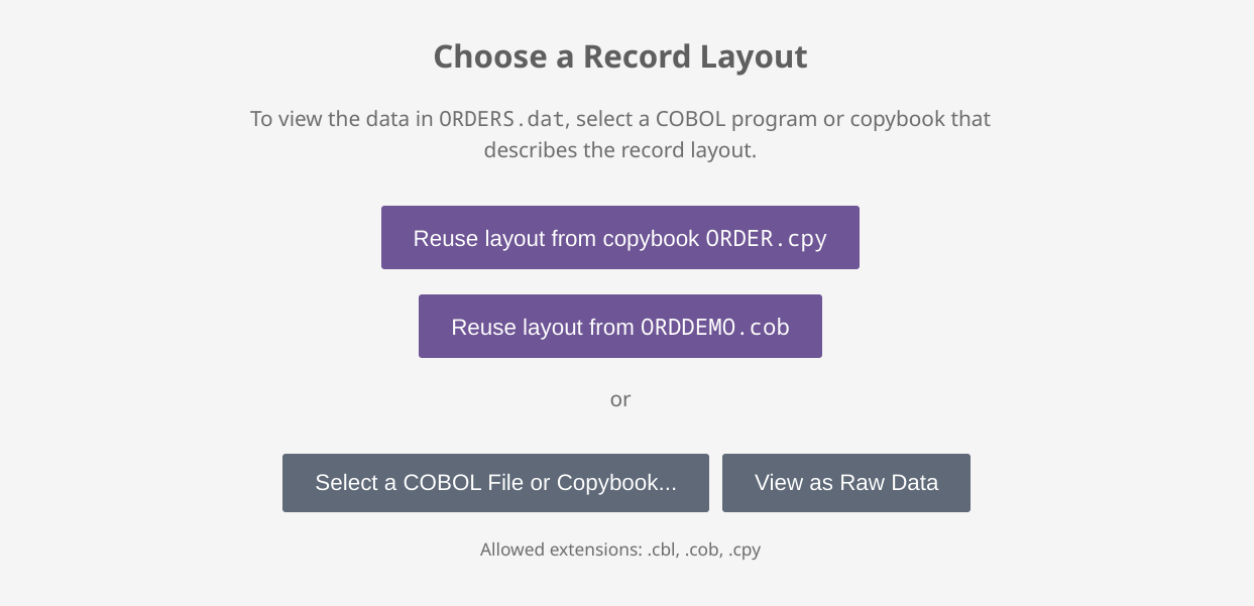


Figure 30: Selection window for the COBOL program or copybook describing the opened data file.

Click on `Select a COBOL File or Copybook...` and select the corresponding file on your computer. The accepted extensions are listed below the buttons. This file can be located in the open project or anywhere else on your disk. The association is saved for subsequent openings of that data file, for the current workspace.

When the data file has already been associated with a COBOL program or a copybook, two buttons `Reuse layout from copybook <copybook>` and `Reuse layout from <program>`, naming the last copybook and the last program used, are shown above, allowing you to reuse the previous association directly.

13.2.2.1 Selecting a Copybook

When a copybook is selected, the workspace is searched for the COBOL programs that include it with a `COPY` statement and whose record layout can be extracted:

- if **a single program** is found, it is used automatically, without any further action on your part;
- if **several programs** are found, a `Multiple programs found` window asks you to choose the one to use from a list (see Selection Lists). The `Select a COBOL File or Copybook...` button also lets you select another file manually;

- if **no program** is found, you are invited to select a COBOL program manually.

It is also possible to view the raw file without a COBOL program by clicking on `View as Raw Data`.

> **Tip**
>
> If you do not have access to the corresponding COBOL program, you can still open the file by clicking on `View as Raw Data` to inspect its raw content.

13.2.3 Step 3: Disambiguation (if necessary)

> **Note**
>
> This step only appears if the selected COBOL program declares multiple files in its `FILE-CONTROL` section and none of them can be identified automatically. The file opens directly if only one file is declared, or if the `ASSIGN` clause of one of the declarations matches the name of the data file.

The same COBOL program can declare multiple files in its `FILE-CONTROL` section. When no declaration that corresponds to the opened file can be automatically determined, a `Multiple file declarations found` disambiguation window is displayed, asking you to choose it.

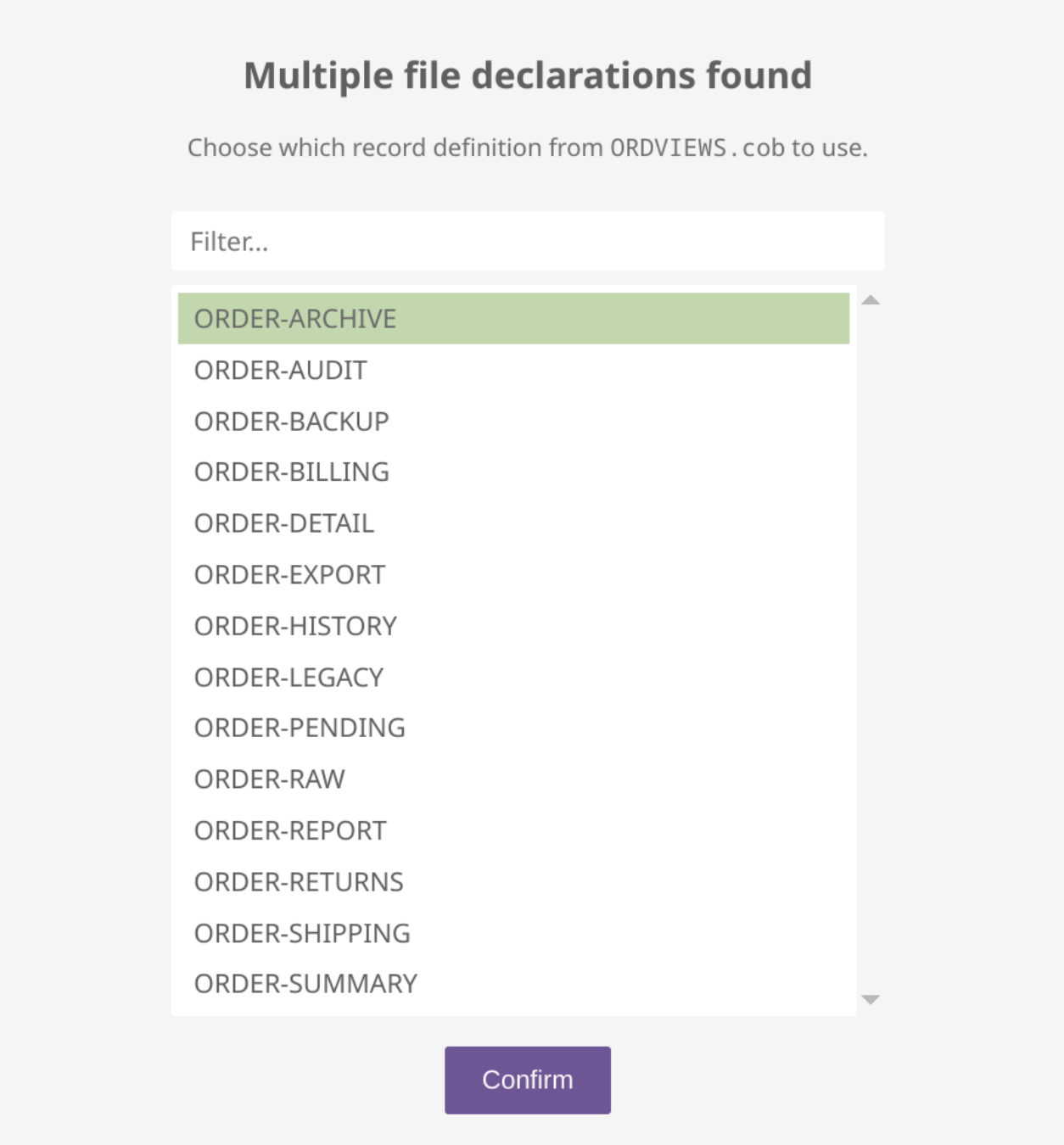


Figure 31: Disambiguation window for file declarations (when multiple data files are declared by the same COBOL program).

Select the appropriate declaration from the list, then click `Confirm` to validate.

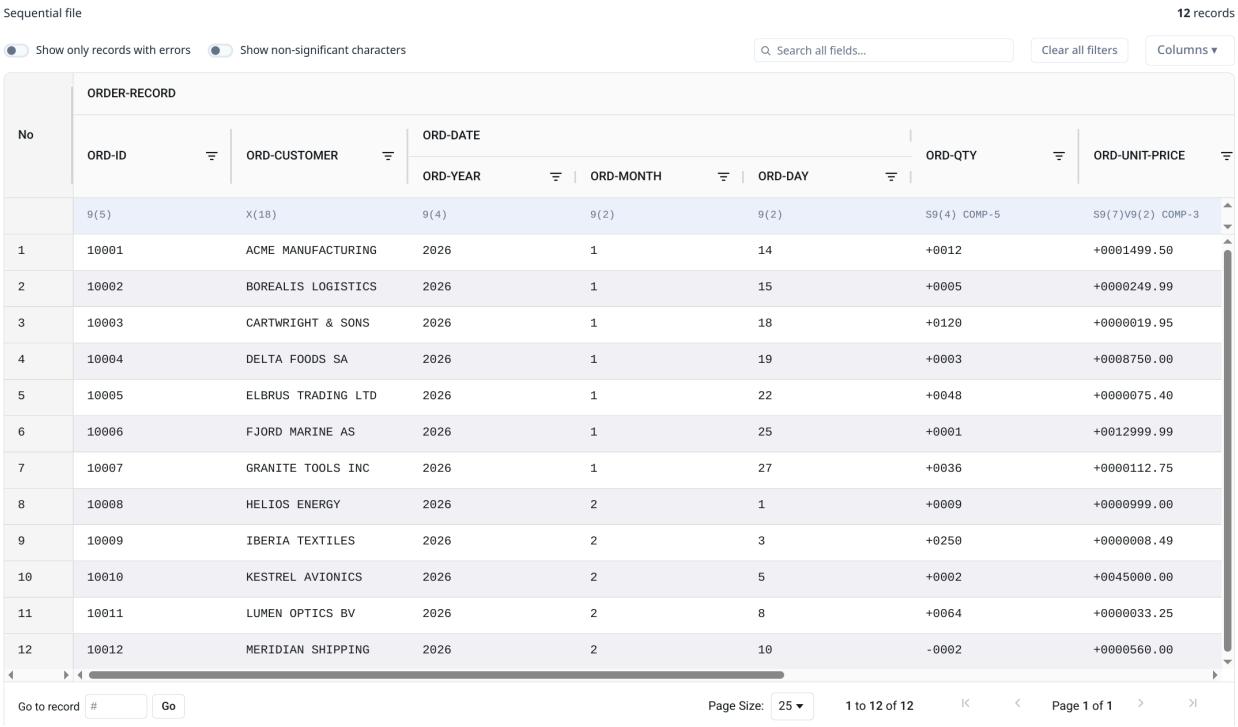
13.2.3.1 Selection Lists

The disambiguation windows, whether for choosing a file declaration or a program including a copybook, display the possible choices in a scrollable list:

- the entries are sorted alphabetically;
- double-clicking an entry selects it and validates the choice directly, without going through the `Confirm` button;
- the last entry chosen for this data file is pinned at the top of the list and pre-selected;
- when the list holds more than ten entries, a `Filter...` search bar placed above it allows you to filter its contents instantly.

13.3 Viewer Interface

Once the file is opened with the associated COBOL program, the viewer displays the structured data in a table format, with one row per record and one column per field (columns can be grouped to represent COBOL data groups).



No	ORDER-RECORD						
	ORD-ID	ORD-CUSTOMER	ORD-DATE			ORD-QTY	ORD-UNIT-PRICE
			ORD-YEAR	ORD-MONTH	ORD-DAY		
	9(5)	X(18)	9(4)	9(2)	9(2)	S9(4) COMP-5	S9(7)V9(2) COMP-3
1	10001	ACME MANUFACTURING	2026	1	14	+0012	+0001499.50
2	10002	BOREALIS LOGISTICS	2026	1	15	+0005	+0000249.99
3	10003	CARTWRIGHT & SONS	2026	1	18	+0120	+0000019.95
4	10004	DELTA FOODS SA	2026	1	19	+0003	+0008750.00
5	10005	ELBRUS TRADING LTD	2026	1	22	+0048	+0000075.40
6	10006	FJORD MARINE AS	2026	1	25	+0001	+0012999.99
7	10007	GRANITE TOOLS INC	2026	1	27	+0036	+0000112.75
8	10008	HELIOS ENERGY	2026	2	1	+0009	+0000999.00
9	10009	IBERIA TEXTILES	2026	2	3	+0250	+0000008.49
10	10010	KESTREL AVIONICS	2026	2	5	+0002	+0045000.00
11	10011	LUMEN OPTICS BV	2026	2	8	+0064	+0000033.25
12	10012	MERIDIAN SHIPPING	2026	2	10	-0002	+0000560.00

Figure 32: Main interface of the file viewer.

From top to bottom, the interface is divided into three main parts where several interactions are possible:

- the header, giving the organization of the file and its total number of records, together with a control panel;
- the data table, where the columns can be moved freely, except for the `No` column of record numbers, which stays pinned to the left, and a field cannot be taken out of the group it belongs to;
- the navigation panel.

The first row of the table is pinned below the column names and gives the `PICTURE` and `USAGE` clauses of each field. Hovering over one of its cells displays additional information about the field, such as whether it is signed and how it is stored.

13.4 Data Visualization

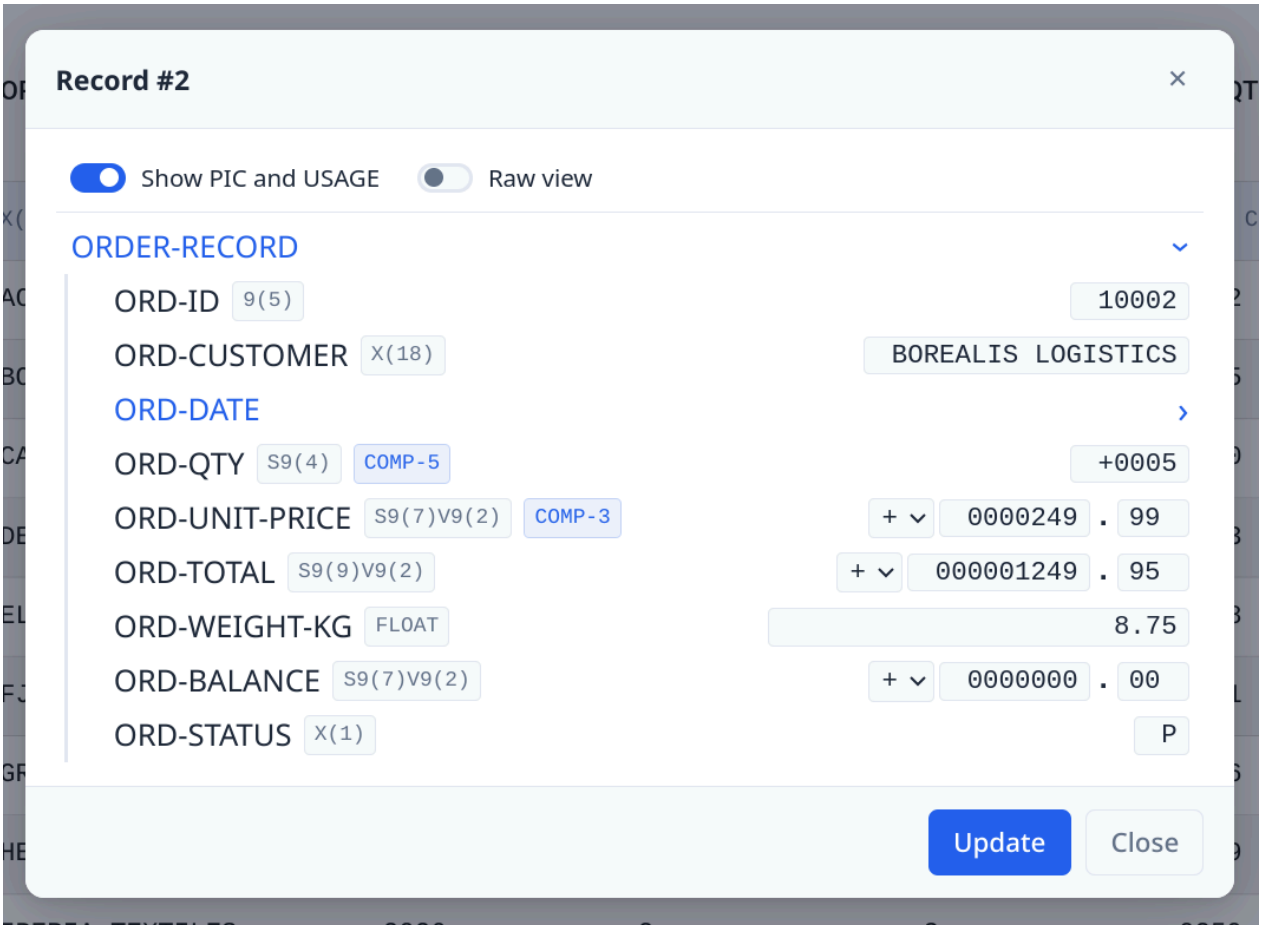


Figure 33: The *record details* window. Each field features labels for its `PICTURE` and `USAGE` clauses, and the associated value on the right-hand side.

By double-clicking a row in the table, the *record details* window, titled `Record #n` where `n` is the record number, opens to display the details of the associated record. The record structure presents the data in a tree format to visually distinguish the different groups. The groups are collapsed when the window opens: clicking on a group name expands it, and clicking it again collapses it.

At the top of the window are formatting buttons:

- a `Show PIC and USAGE` button allows you to show or hide labels, which helps visually lighten the interface;

- a `Raw view` button allows you to display the values as they are stored on disk, with their non-significant characters: for text values, leading and trailing spaces are shown as middle dots (`·`), and for numeric values the leading zeros are dimmed. Signs are also displayed at the position where they are stored, at the beginning or at the end of the value. Fields cannot be edited in this mode.

When the length of a record does not match the one declared by the record layout, it is marked in the `No` column of the table as well as in the *record details* window. The fields located past the end of the record then hold no data, and the record cannot be edited.

13.5 Editing a Record

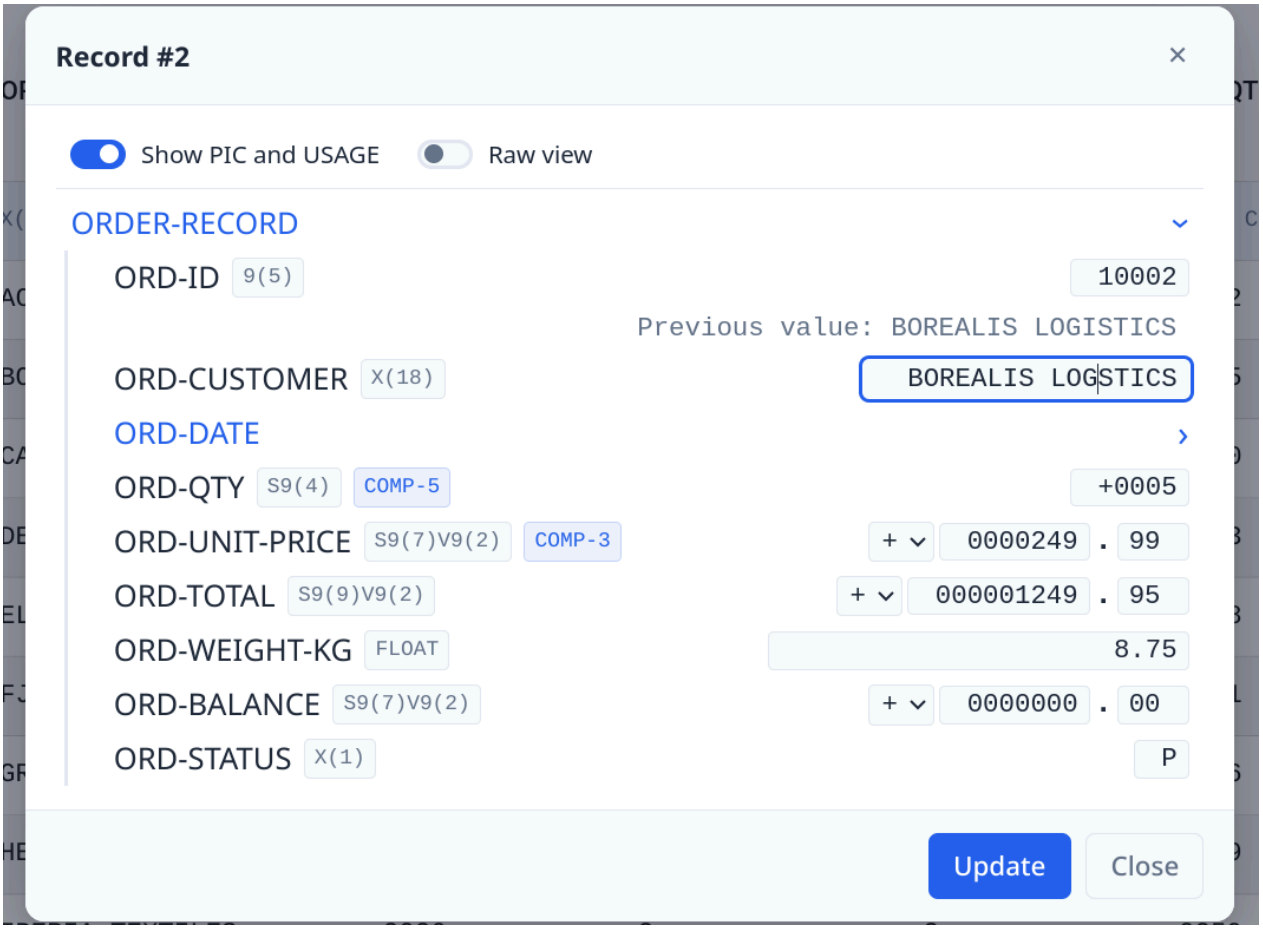


Figure 34: Editing a field: the previous value is recalled above the input.

Outside of the `Raw view` mode, the values displayed in the *record details* window are editable input fields. Numeric values with a decimal part are split into a sign selector (`+` / `-`) for signed fields, an integer part and a decimal part.

As soon as a field is modified, its previous value is recalled above the input (`Previous value: ...`).

Each input is checked against the `PICTURE` clause of the field (number of digits, presence of a decimal part, allowed characters, etc.). In case of error, an explanatory message is displayed next to the field and the `Update` button is disabled until the error is fixed.

Modifications are only saved when clicking `Update`: the `Close` button closes the window without saving pending changes. Clicking `Update` writes only the modified fields to the data file, the other ones keeping their original content. The window stays open and a message lists the modified fields (`Saved. Changed fields: ...`), while the table of the main window is updated.

> ⚠ Warning
>
> Changes are written directly to the data file. Remember to keep a backup copy if needed.

13.6 Formatting



Figure 35: Buttons for formatting record values.

At the top left of the main window are buttons to format the data.

Similarly to the *record details* window, you can show or hide non-significant characters using the `Raw view` toggle. The same setting applies in both windows: changing it in one of them changes it in the other. Like the column selection, the state of this toggle and that of `Show PIC and USAGE` are kept for subsequent openings of the viewer.

With the `Show only records with errors` toggle, you can also filter the view to keep only records containing errors (notably typing errors, when values do not match the specified `PICTURE` clause). This makes it easy to spot data that needs fixing. Records containing errors are highlighted in the table at all times, whether this toggle is enabled or not.

13.7 Data Navigation



Figure 36: Navigation interface for records at the bottom of the main window.

At the bottom of the main window is a navigation interface. In order:

- a `Go to record` field allowing you to jump to a specific record number. For example, entering `5` and clicking `Go` will take you directly to the fifth record (which will be highlighted). The `Enter` key validates the input as well, and a record number outside the range of the file is ignored;

- a `Page Size` drop-down list allowing you to choose the number of records displayed per page;
- an indicator of the currently displayed record numbers relative to the total (`1 to 10 of 10` in the screenshot);
- an indicator of the current page relative to the total number of pages (`Page 1 of 1` in the screenshot), surrounded by buttons to go to the next / previous page or to the beginning / end of the file.

13.8 Filtering

13.8.1 Column Set Filtering

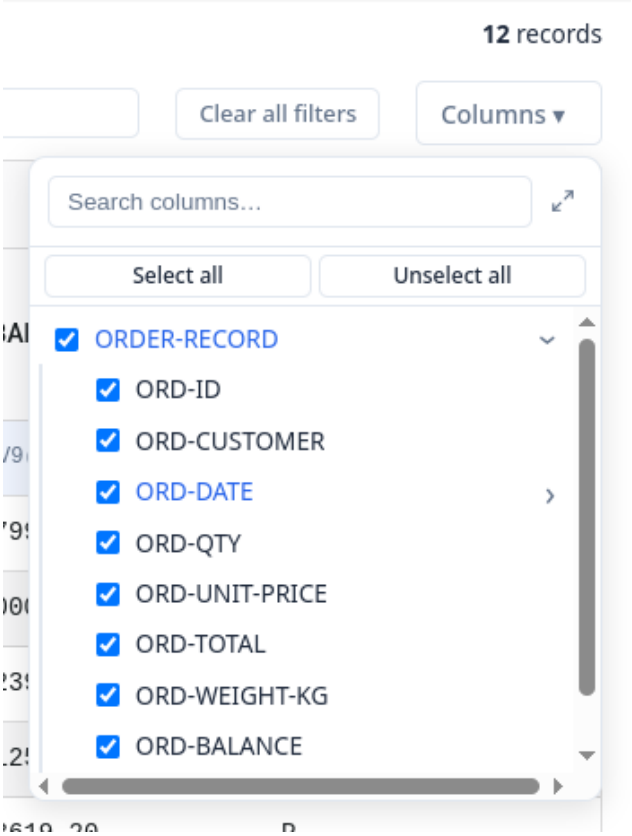


Figure 37: Column display management panel.

At the top right of the main window is a `Columns` button that opens a floating panel with the list of fields. The list follows the structure of the record: the fields of a group appear indented under its name.

This allows you to select which columns are displayed or hidden by checking or unchecking the corresponding boxes. Checking or unchecking a group applies to all of its fields at once,

and the box of a partially displayed group is shown in an intermediate state. `Select all` and `Unselect all` buttons are also present to select and deselect all columns at once.

Unchecking an `OCCURS` group also hides the column displayed when it is folded, so that the group disappears entirely from the table.

The selection is kept for subsequent openings of the viewer, and the number of hidden columns is displayed next to the `Columns` button.

A search bar at the top of the panel allows you to search for specific columns.

> **Tip**
>
> If you only want to display a small number of specific columns, you can uncheck everything with `Unselect all` and use the search bar to find and check the columns you need.

A "fullscreen" button at the top right of the floating window also allows for a wider and more comfortable view in case there is a great depth of data.

13.8.2 Column Data Filtering

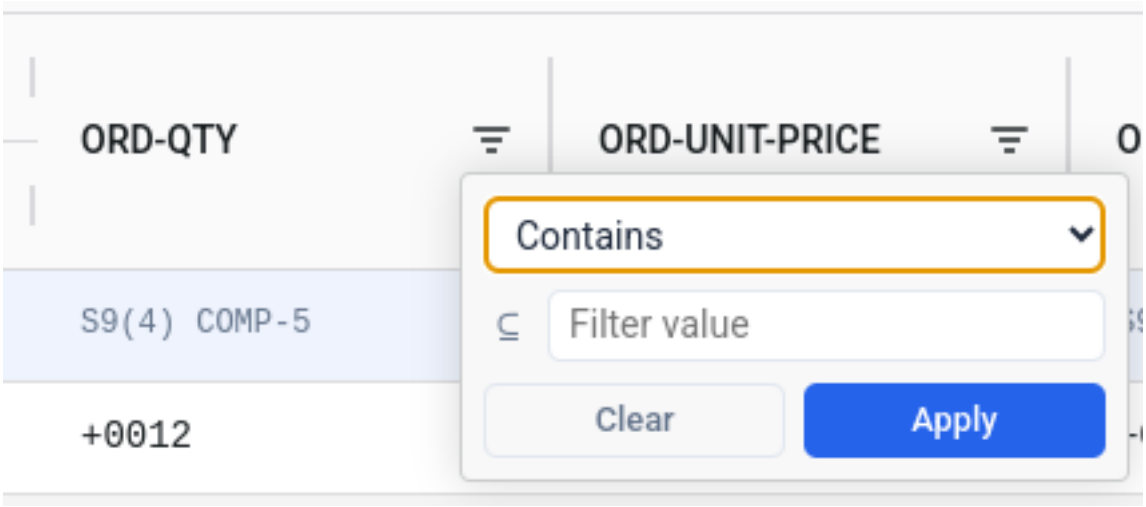


Figure 38: Floating window for filtering column data.

Each column has a "funnel" icon to the right of its name, allowing you to filter the displayed data according to certain criteria.

Whatever the type of the field, the filters are:

- `Contains` to keep only records where the value in the selected column contains the entered text;
- `Starts with` when the value starts with the input;
- `Ends with` when the value ends with the input;
- `Equals` when the value is identical to the input;

- `Blank` when the value is empty;
- `Not blank` when the value is not empty.

Comparisons are case-sensitive. The filter is applied by clicking `Apply`, and `Clear` removes it.

13.8.3 Global Search



Figure 39: Global search bar accompanied by a button to reset all filters.

A search bar is available at the top right of the main window. It allows you to search for a value across all columns of the data file (in the same way you would manually search for a keyword in the filter associated with each column): a record is kept as soon as one of its fields contains the searched text, again case-sensitively. Pressing the `Enter` key validates the search.

The bar is followed by a `Clear all filters` button that resets all filters, including both global search and column-specific filters.

13.9 Tables (OCCURS clause)

A group with an `OCCURS n` clause is displayed as a table of columns called “occurrences”, indexed by an integer.

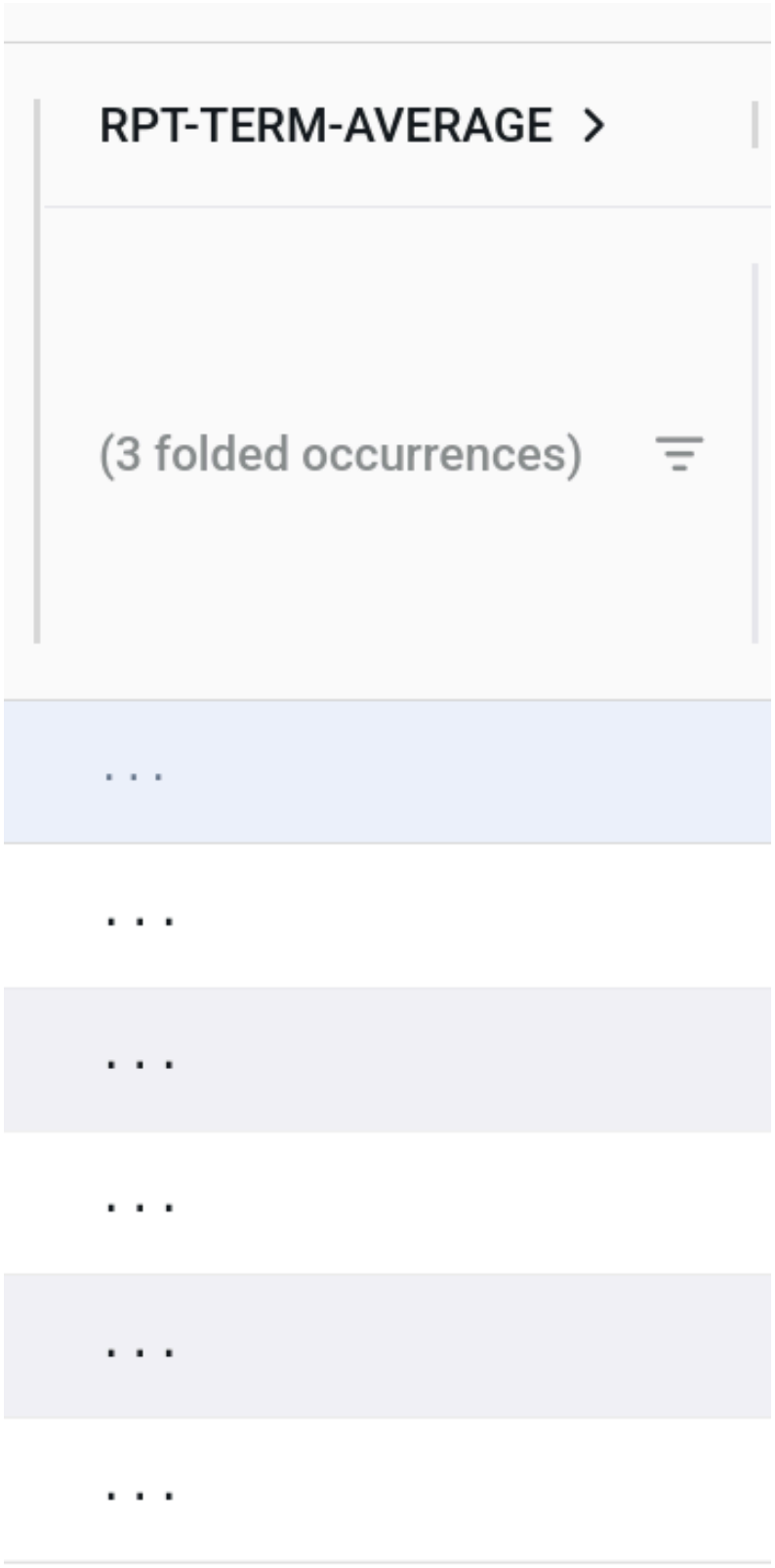
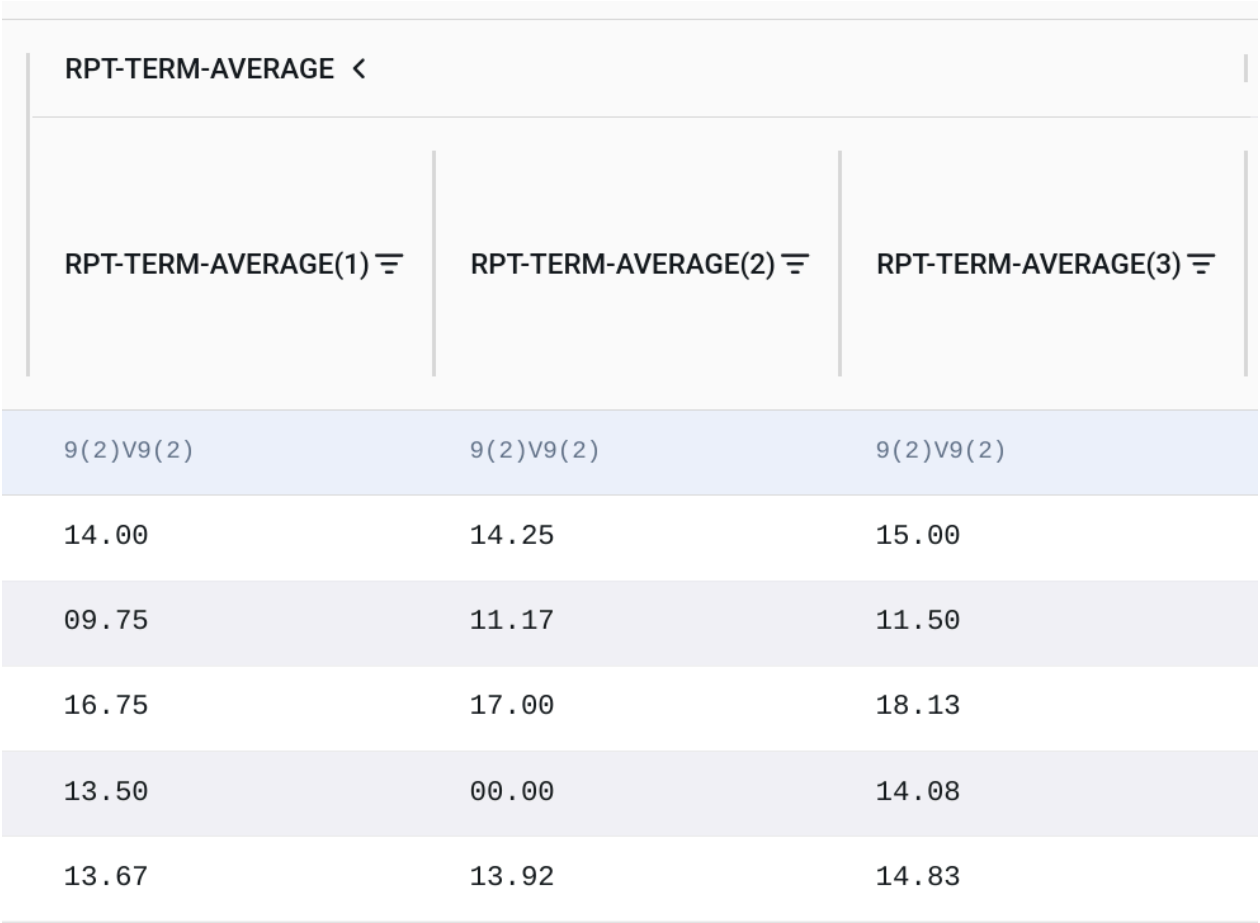


Figure 40: Folded OCCURS table: a single greyed-out column summarizes the occurrences.

When the file is opened, this table is folded and displays a single greyed-out column, titled `(n folded occurrences)` where `n` is the number of occurrences. The cells in this column show "`...`". Therefore, the table remains readable when a record repeats a large number of fields.



RPT-TERM-AVERAGE		
RPT-TERM-AVERAGE(1)	RPT-TERM-AVERAGE(2)	RPT-TERM-AVERAGE(3)
9(2)V9(2)	9(2)V9(2)	9(2)V9(2)
14.00	14.25	15.00
09.75	11.17	11.50
16.75	17.00	18.13
13.50	00.00	14.08
13.67	13.92	14.83

Figure 41: The same `OCCURS` table unfolded: one column per occurrence.

Clicking anywhere on the group header unfolds it, and clicking again folds it back. Once unfolded, the group replaces that single column with one column per occurrence.

The headers of these columns repeat the field name followed by its occurrence index. The occurrences remain side by side and in the order of their indices when the columns are moved.

RPT-SUBJECT				
RPT-SUBJECT(1)			RPT-SUBJECT(2)	
SUB-NAME	SUB-TERM-GRADE (3 folded occurrences)	SUB-AVERAGE	SUB-NAME	SUB-TERM-GRADE (3 folded occurrences)
X(10)	...	9(2)V9(2)	X(10)	...
MATHS	...	15.25	FRENCH	...
MATHS	...	09.92	FRENCH	...
MATHS	...	18.08	FRENCH	...
MATHS	...	09.08	FRENCH	...
MATHS	...	12.50	FRENCH	...

Figure 42: Nested `OCCURS` tables.

In case of nested `OCCURS`, each occurrence of the outer group has its own copy of the inner group, which is folded when the file is opened as well. Each copy folds and unfolds independently of the others.

RPT-SUBJECT				
RPT-SUBJECT(1)				
SUB-NAME	SUB-TERM-GRADE			SUB-AVERAGE
	SUB-TERM-GRADE(1)	SUB-TERM-GRADE(2)	SUB-TERM-GRADE(3)	
X(10)	9(2)V9(2)	9(2)V9(2)	9(2)V9(2)	9(2)V9(2)
MATHS	14.50	15.00	16.25	15.25
MATHS	08.50	10.00	11.25	09.92
MATHS	17.00	18.25	19.00	18.08
MATHS	13.25	00.00	14.00	09.08
MATHS	11.75	12.25	13.50	12.50

Figure 43: The inner `OCCURS` table unfolded in the first occurrence of the outer table.

The filter of an occurrence column applies only to that occurrence. On the other hand, the global search, that covers all the columns, applies to every occurrence.

In the *record details* window, the field names keep their occurrence index.

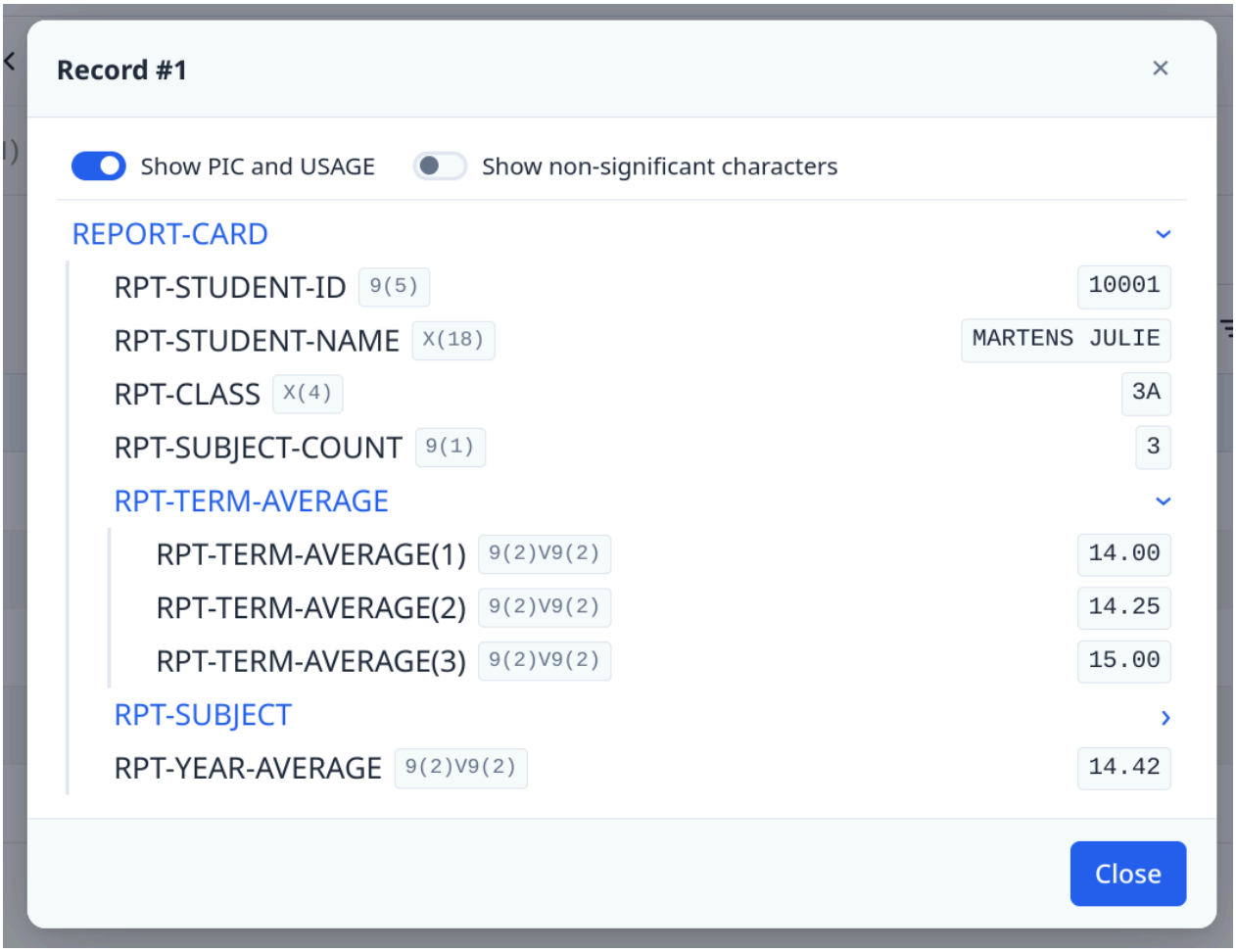


Figure 44: The *record details* window: the occurrences keep their index there.

14 SuperBOL Studio Reference

14.1 SuperBOL Studio Configuration

There are several configuration files used by VSCode:

- By default, options are taken from the user configuration in `$HOME/.config/Code/User/settings.json` (on Linux)
- In a workspace, options are taken from the workspace configuration in `.vscode/settings.json`
- Additionnally, SuperBOL Studio LSP server will also use options from `superbol.toml` at the root of the workspace, as a preference over options from `settings.json` files. This file is also used by other tools from SuperBOL that are not related to VSCode.

The command **SuperBOL: Write Project Configuration** can be used to export the configuration from the `settings.json` to the `superbol.toml` file.

14.2 SuperBOL options in `settings.json`

- **superbol.cobol.dialect** (string, default: `default`): Default COBOL dialect for all files

  The available dialects are:

  - **default** and **gnucobol**: a global dialect including all other dialects
  - **cobol85**: only keywords from ANSI COBOL 85
  - **cobol2002**: only keywords from COBOL ISO 2002
  - **cobol2014**: only keywords from COBOL ISO 2014
  - **acu** and **acu-strict**: the ACUCOBOL-GT dialect
  - **bs2000** and **bs2000-strict**: dialect for Fujitsu's COBOL2000 V1.5 compiler
  - **gcos** and **gcos-strict**: dialect for COBOL on BULL GCOS mainframes
  - **ibm** and **ibm-strict**: dialect for COBOL on z/OS IBM mainframes
  - **mf** and **mf-strict**: dialect for Visual COBOL by MicroFocus (RocketSoftware now)
  - **mvs** and **mvs-strict**: dialect for IBM COBOL for MVS & VM
  - **realia** and **realia-strict**: dialect for CA Realia II
  - **rm** and **rm-strict**: dialect for RM-COBOL
  - **xopen**: dialect for X/Open COBOL

The `-strict` version of a dialect permits using reserved words from all other dialects as custom COBOL words, as the non-strict version also reserve these words.

You may also specify an absolute path to a custom GnuCOBOL dialect configuration file (see `cobc`'s `-conf` option).

(May be overriden in project-specific `superbol.toml` files)

- **superbol.cobol.sourceFormat** (string, default: `auto`): Default source reference-format.

  The available formats are:

  - **auto**: SuperBOL will try to discover the format by automatically
  - **fixed**: the old format, with a non-significant left margin of 6 characters, an indicator (`*` for comments) and a non-significant right margin after column 72
  - **cobol85**: same as `fixed`, but with strict verification of Area A.
  - **variable**: Same as `fixed`, but non-significant right margin starts after column 252
  - **xcard**: ICOBOL xCard format. Same as `fixed`, but non-significant right margin starts after column 255.
  - **free**: the new free format, with no margins and limitations. Comments are introduced by `*>` (except in MicroFocus dialect, where a `*` should be put in column 1)
  - **xopen**: X/Open Free format. Text starts at column 1, unless an indicator is present (`*` for comments, `D` for debug), and lines of up to 80 characters.
  - **crt**: ICOBOL Free-form format. Same as `xopen`, but with lines of up to 320 characters.
  - **terminal**: ACUCOBOL-GT Terminal format. Same as `crt`. Mostly compatible with VAX COBOL terminal format.
  - **cobolx**: Indicator in column 1, text starts at column 2, and ends at column 255.

  (May be overriden in project-specific `superbol.toml` files)

- **superbol.cobol.copybooks** (array, default: `[ { "dir: "." }]`): Copybook Paths. Each object in the path contains at least a directory in the `dir` field, and a `file-relative` field (default it `true`).

  (May be overriden in project-specific `superbol.toml` files)

- **superbol.cobol.copyexts** (array, default: `["cpy","cbl","cob"]`): File extensions for copybook resolution

  (May be overriden in project-specific `superbol.toml` files)

- **superbol.cobcPath** (string, default: “cobc”): GnuCOBOL Compiler Executable: Path to the GnuCOBOL compiler executable

- **superbol.lspPath** (string, default: “”): SuperBOL Executable

  Name of the `superbol-free` executable if available in PATH; may be an absolute path otherwise. Leave empty to use the bundled `superbol-free`, if available.

- **superbol.forceSyntaxDiagnostics** (boolean, default: false): Force reporting of syntax diagnostics for dialects other than `COBOL85`
- **superbol.cacheInGlobalStorage** (boolean, default: false): Use storage provided by Visual Studio Code for caching. When this setting is set to *false*, the cache related to the contents of a given workspace folder *f* is stored in a file named *f*`/_superbol/lsp-cache`.: cache files are not removed automatically, whatever their location.
- **superbol.debugger.displayVariableAttributes** (boolean, default: false): Display Variable Attributes

  Display storage property and field attributes (e.g. size of alphanumerics, digits and scale of numerics).

- **superbol.debugger.libcobPath** (string, default: null): GnuCOBOL Runtime Library

  Path to the GnuCOBOL runtime library file

- **superbol.debugger.gdbPath** (string, default: “gdb”): GNU Debugger Executable

  Path to the GNU debugger executable.

- **superbol.debugger.cobcrunPath** (string, default: “cobcrun”): GnuCOBOL module loader

  Path to `cobcrun`, the GnuCOBOL module loader.

14.3 SuperBOL Options in `superbol.toml`

Not all options from `settings.json` can be set in `superbol.toml`. The following options can be set, in the `[cobol]` section of the TOML file:

- **`dialect`**: Default dialect for COBOL source files
- **`source-format`**: Default source reference-format
- **`copybooks`**: Where to find copybooks
- **`copyexts`**: Copybook filename extensions (for example `["cpy", "cob"]`)

14.4 SuperBOL Commands

- **SuperBOL: Restart Language Server**:
- **SuperBOL: Write Project Configuration**: translate options from the `settings.json` file to the `superbol.toml` file.
- **SuperBOL: Show Coverage**:
- **SuperBOL: Hide Coverage**:

- **SuperBOL: Update Coverage**:
- **SuperBOL: Show Control-flow**:
- **SuperBOL: Show Control-flow as an Arc-diagram**:

15 Troubleshooting