# COBOL-IT® Developer Studio
# Getting Started
# The Debugger Perspective
# Version 2.0

# Acknowledgment

**Contact Information:**

COBOL-IT
231, rue Saint-Honoré - 75001 Paris - FRANCE
Tel.: +33 1 75 43 05 50
Fax: +33 1 75 43 05 16
Email: contact@cobol-it.com
www.cobol-it.com

# COBOL-IT Developer Studio Topics

## Introduction

This document describes how to install and how to use the **COBOL-IT Developer Studio**, which is COBOL-IT's eclipse-based development environment, designed to support users of the **COBOL-IT Compiler Suite.** COBOL-IT is based on OpenCOBOL, originally developed by Keisuke Nishida and maintained since 2007 by Roger While. In 2008, COBOL-IT forked its own compiler branch, with the intention of developing a fully featured product and offering professional support to the COBOL user industry.

## COBOL-IT Developer Studio License terms

For more information, please contact us at: contact@cobol-it.com

COBOL-IT Corporate Headquarters are located at

231, rue Saint-Honore
75001 Paris
Tel: +33.1.75.43.05
Email: contact@cobol-it.com

# Dependencies

Dependencies for the Developer Studio are:

| Dependency | Comment |
|---|---|
| "C" compiler | The COBOL-IT Compiler requires a "C" compiler. While most Linux>Unix installations will include a "C" compiler, many Windows installations will not. Windows users can download the Visual Studio from www.microsoft.com. |
| COBOL-IT Compiler Suite | The COBOL-IT Compiler Suite, Standard Edition can be downloaded at the COBOL-IT Online Portal. For access to the COBOL-IT Online Portal, please contact your sales representative at sales@cobol-it.com. |
| Java Runtime Environment (JRE) | The COBOL-IT Developer Studio Kepler build can be run with the Java Runtime Environment (JRE) Version 1.6 or greater. The COBOL-IT Developer Studio Neon build can be run with the JRE Version 1.8 or greater. |
| Eclipse | Eclipse is included with the download of Developer Studio. |

## The COBOL-IT Developer Studio Distribution

For Windows-based installations, the COBOL-IT Developer Studio, Enterprise Edition can be downloaded  from the COBOL-IT online portal with a login and password provided by your sales representative.

The COBOL-IT Developer Studio, Enterprise Edition is available with Subscription. The COBOL-IT Developer Studio, Enterprise Edition provides functionality with the installation of several Perspectives:

- Developer Studio Perspective in which users set up and build COBOL projects, using a locally installed version of the COBOL-IT Compiler Suite Enterprise Edition. The Developer Studio Perspective additionally provides access to Code Coverage and Profiling Tools.
- Debugger Perspective providing access to a feature-rich COBOL debugger both locally, and on Remote Systems
- Remote Systems Perspective, allowing use of Compiler, Runtime, and Debugger functionalities installed on remote servers.

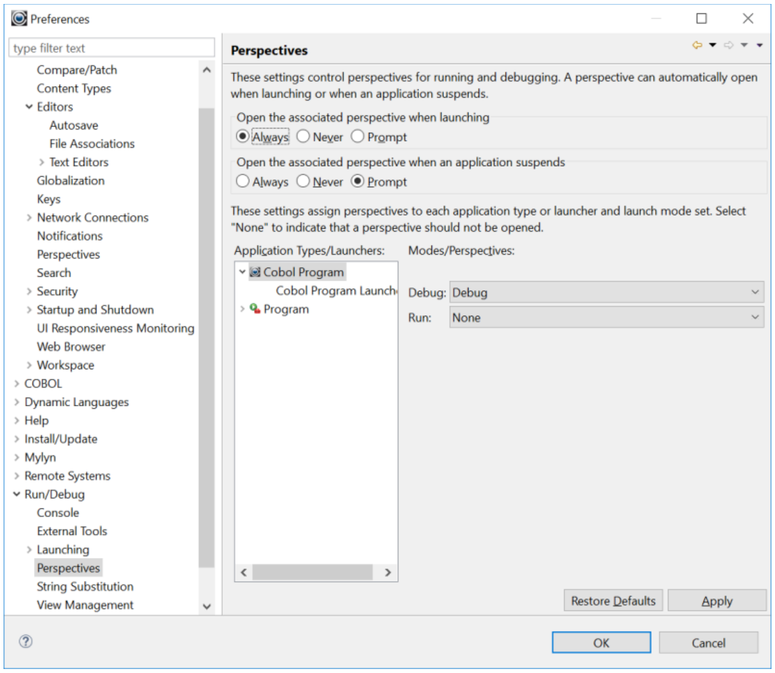
- Git and RSEGit Perspectives, providing users with full access to the Git/Github Source Code Control System.
- Data Displayer Perspective, providing access to a tool for browsing and modifying data in indexed, sequential and relative files.
- Planning Perspective, providing access to the Mylyn Task Manager.
- For more information about the usage of Git/RSEGit, Data Displayer, Mylyn Task Manager, and Code Coverage, see the Getting Started with the Developer Studio- The Utilities Manual.
- Using the COBOL-IT Developer Studio requires a license for both the COBOL-IT Compiler Suite Enterprise Edition, and COBOL-IT Developer Suite.

# The Debugger Perspective

## Configuration of the Debugger Perspective

### Window>Preferences>Run/Debug>Perspectives

Set “Open the associated perspective when launching” to “Always”. This will have the effect of causing the Debugger Perspective to open when clicking on the Debug toolbar button, or pressing F11.



Perspectives

These settings control perspectives for running and debugging. A perspective can automatically open when launching or when an application suspends.

Open the associated perspective when launching
● Always ○ Never ○ Prompt

Open the associated perspective when an application suspends
○ Always ○ Never ● Prompt

These settings assign perspectives to each application type or launcher and launch mode set. Select "None" to indicate that a perspective should not be opened.

Application Types/Launchers: Cobol Program > Cobol Program Launch

Modes/Perspectives:
Debug: Debug
Run: None

## Window>Preferences>General>Keys

Debugger Hot Keys can be configured in the **Window>Preferences>General>Keys** interface. Pre-set Debugger Hot Keys include:

| Debugger Function | Hot Key | Usage |
|---|---|---|
| Step Into | F5 | The Step Into function single steps through lines of code. The Step Into function enters paragraphs/subprograms that are the targets of PERFORM / CALL statements. |
| Step Over | F6 | The Step Over function can be executed when positioned on a PERFORM or CALL statement. The Step Over Into function causes paragraphs/subprograms that are the targets of PERFORM / CALL statements to be executed, and then Steps to the next line of code. |
| Step Return | F7 | The Step Return function can be executed when positioned inside a paragraphs/subprogram that is the target of a PERFORM / CALL statement. When executed, the rest of the code in the paragraph/section/subprogram is executed, and the program returns to the next line after the PERFORM / CALL statement. |
| Resume | F8 | Resumes the execution of the program. |
| Debug | F11 | Runs a source file in Debug mode. Select a program, and click on the Debug toolbar button or press F11. The Debugger Perspective is launched, a console window is opened, and execution is suspended on the first line of code in the procedure division, which is marked in the COBOL-IT Program View with an arrow indicator. |
| Terminate | Ctrl+F2 | Terminates the program being debugged. |
| Run | Ctrl+F11 | Runs the program, skipping all breakpoints. |
| Toggle Breakpoint | Ctrl+Shift+B | Sets a breakpoint where one does not previously exist, or removes a breakpoint where one does previously exist. Press Ctrl+Shift+B to either create a breakpoint, or remove an existing breakpoint. The Toggle Breakpoint function causes the Breakpoints View to be updated. |

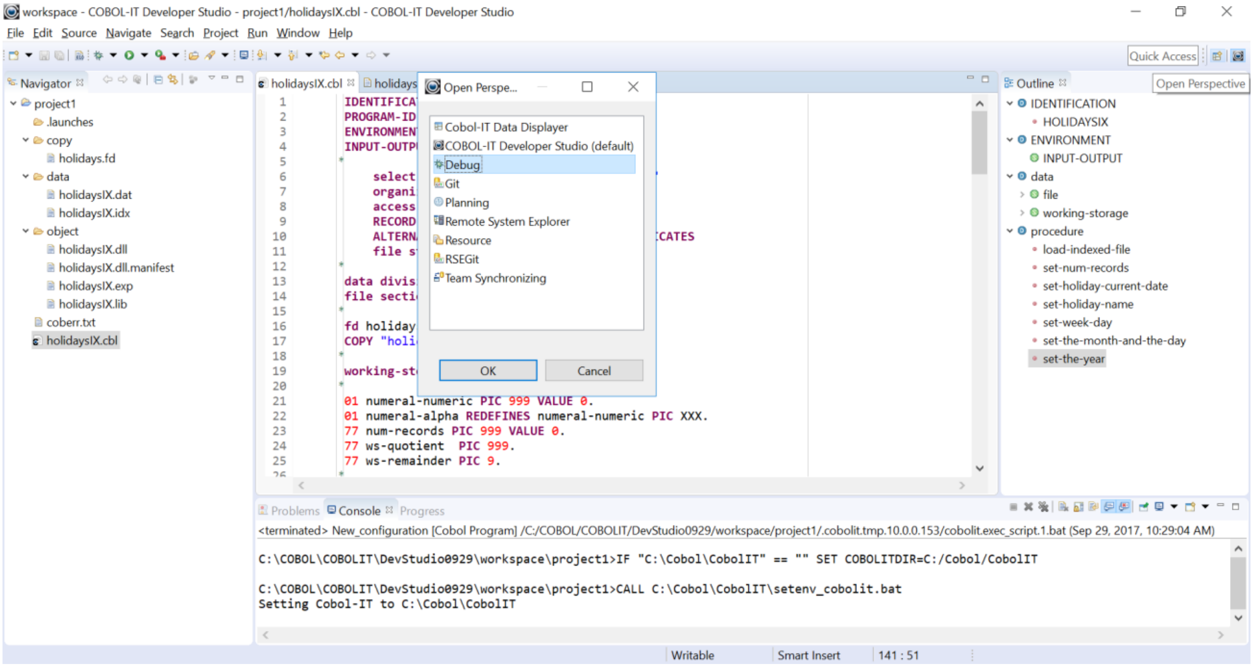
# Important Debugging Operations

## Opening the Debugger Perspective

### *Open Perspectives Tab*

The Debugger Perspective can be opened through the Open Perspectives dialog window.

Click on the Open Perspectives push-button in the upper-right corner of the Developer Studio, select Other…, and then click on COBOL-IT Debugger in the Open Perspective Window. This opens the Debugger Perspective directly.



```
       IDENTIFICA
       PROGRAM-ID
       ENVIRONMEN
       INPUT-OUTP
      *
           select
           organi
           access
           RECORD
           ALTERN                                    CATES
           file s
      *
       data divis
       file secti
      *
       fd holiday
       COPY "holi
      *
       working-st
      *
       01 numeral-numeric PIC 999 VALUE 0.
       01 numeral-alpha REDEFINES numeral-numeric PIC XXX.
       77 num-records PIC 999 VALUE 0.
       77 ws-quotient  PIC 999.
       77 ws-remainder PIC 9.
      *
```

```
<terminated> New_configuration [Cobol Program] /C:/COBOL/COBOLIT/DevStudio0929/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2017, 10:29:04 AM)

C:\COBOL\COBOLIT\DevStudio0929\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio0929\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

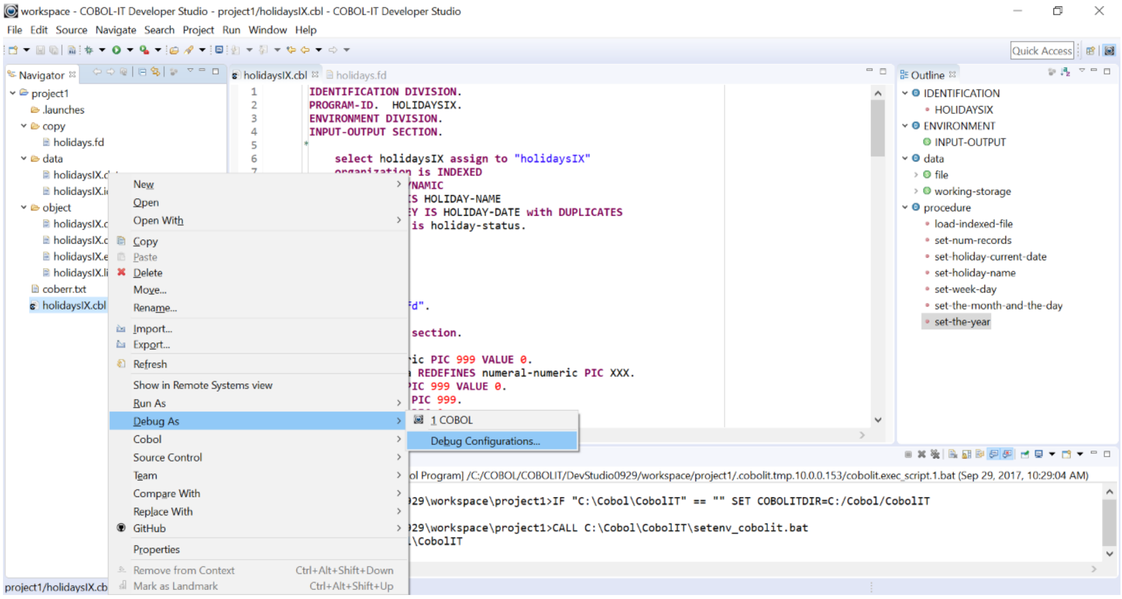
### *Press the Debug button, or press Fn+F11*

Select a source file that has been set up in the project, and press the Debug toolbar button, or hold down the Fn key and press F11. Note that by default, the Debug function is associated with the F11 key in the **Window>Preferences>General>Keys** interface, as described above.

The Debug function uses the default compiler flags associated with the program to compile it, if a Build is required, and then uses the runtime configuration described for the program to set the appropriate environment variables. Pressing the Debug button or pressing Fn+F11 then will initiate the execution of the selected program in the Debugger Perspective.

### *Use the Debug As function*

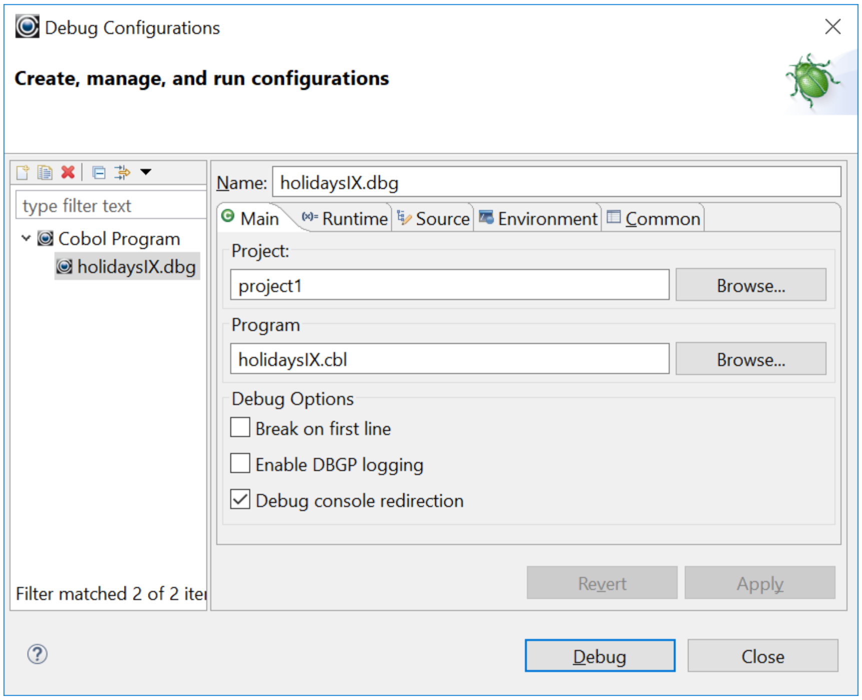
The Debug As function is similar to the Run As function. A Debug Configuration can be configured to contain environment variables needed for debugging purposes.



```
IDENTIFICATION DIVISION.
PROGRAM-ID.  HOLIDAYSIX.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
*
    select holidaysIX assign to "holidaysIX"
    organization is INDEXED
                  NAMIC
                  S HOLIDAY-NAME
                  Y IS HOLIDAY-DATE with DUPLICATES
                  is holiday-status.

                  d".

                  section.

                  ic PIC 999 VALUE 0.
                  REDEFINES numeral-numeric PIC XXX.
                  PIC 999 VALUE 0.
                  PIC 999.
```

```
ol Program] /C:/COBOL/COBOLIT/DevStudio0929/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2017, 10:29:04 AM)

29\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

29\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
\CobolIT
```

We name our new Debug Configuration “holidaysIX.dbg” and associate our new Debug Configuration with the program “holidaysIX.cbl”.



Name: holidaysIX.dbg

Project: project1

Program: holidaysIX.cbl

Debug Options

□ Break on first line

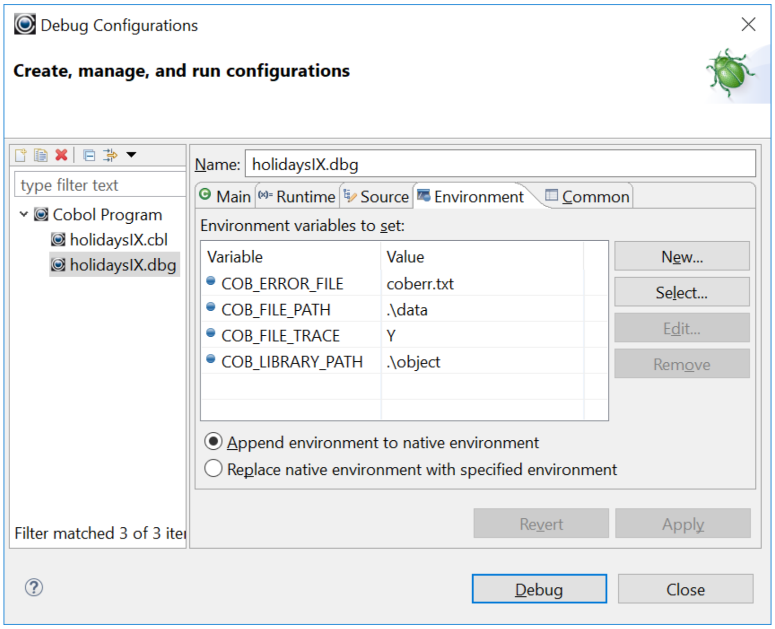
□ Enable DBGP logging

☑ Debug console redirection

The Runtime tab contains default runtime settings.  No changes are necessary.

The Source tab is useful if the debugger needs to locate your source file.  No changes are necessary.

The Environment tab allows you to set runtime environment variables that are only useful in debugging situations.  In the case below, we can see that COB_FILE_TRACE is set to Y, for example.  To use this debug configuration, click Apply, and then click Debug.



Name: holidaysIX.dbg

Environment variables to set:

| Variable | Value |
|---|---|
| COB_ERROR_FILE | coberr.txt |
| COB_FILE_PATH | .\data |
| COB_FILE_TRACE | Y |
| COB_LIBRARY_PATH | .\object |

● Append environment to native environment
○ Replace native environment with specified environment

## The Debug View

In the image below:

- holidaysIX.dbg is the name of the debug configuration

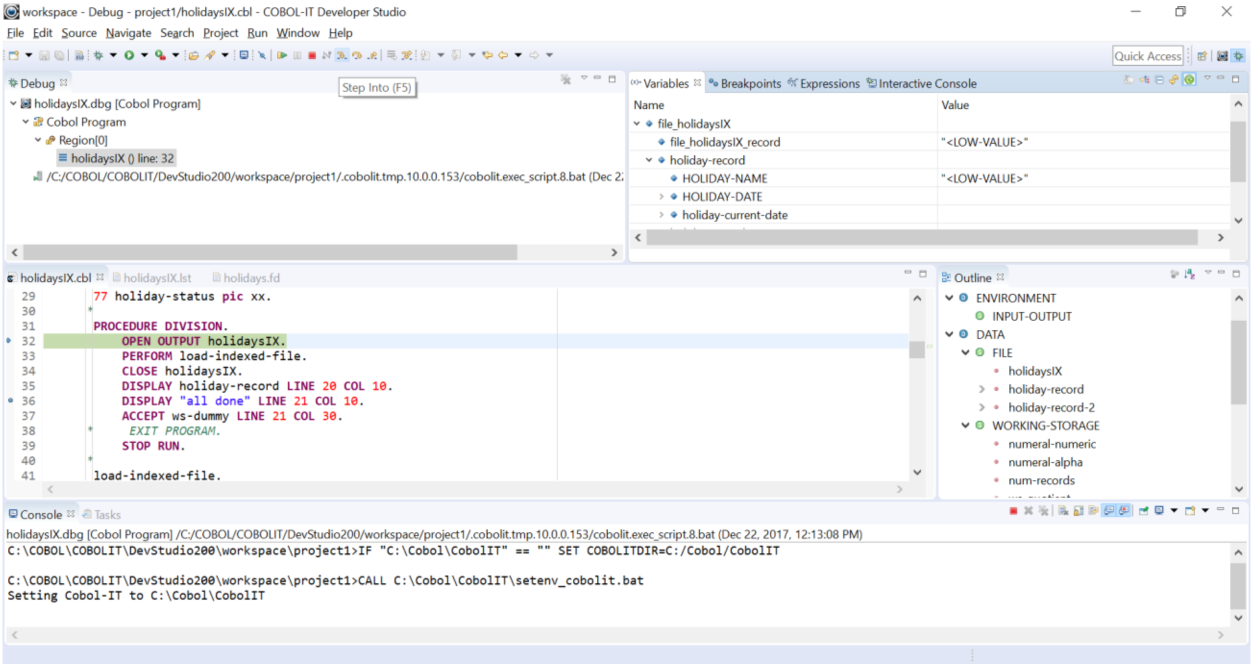
- COBOL Program is the class of program running
- Region(0) is the first thread running in the Program
- HOLIDAYSIX () line 32 represents the compiled object/current line of execution
- C:/COBOL/COBOLIT…./eclipse.exec_script1.bat is the name of the batch file executing the debugger process.

When focus is on the compiled object, as in the image above,  all of the debug functions on the toolbar are enabled.  In this mode, user steps/runs through the program, stopping at breakpoints.  Note that the current line of execution is recorded in the Debug View Window.

## Step/Run/Terminate Functions

### *Single-Step*

With the cursor positioned on a line of code, use the Single-Step/Fn+F5 function to advance to the next line of code. In this case, the cursor will then be positioned on line:
33: PERFORM load-indexed-file.



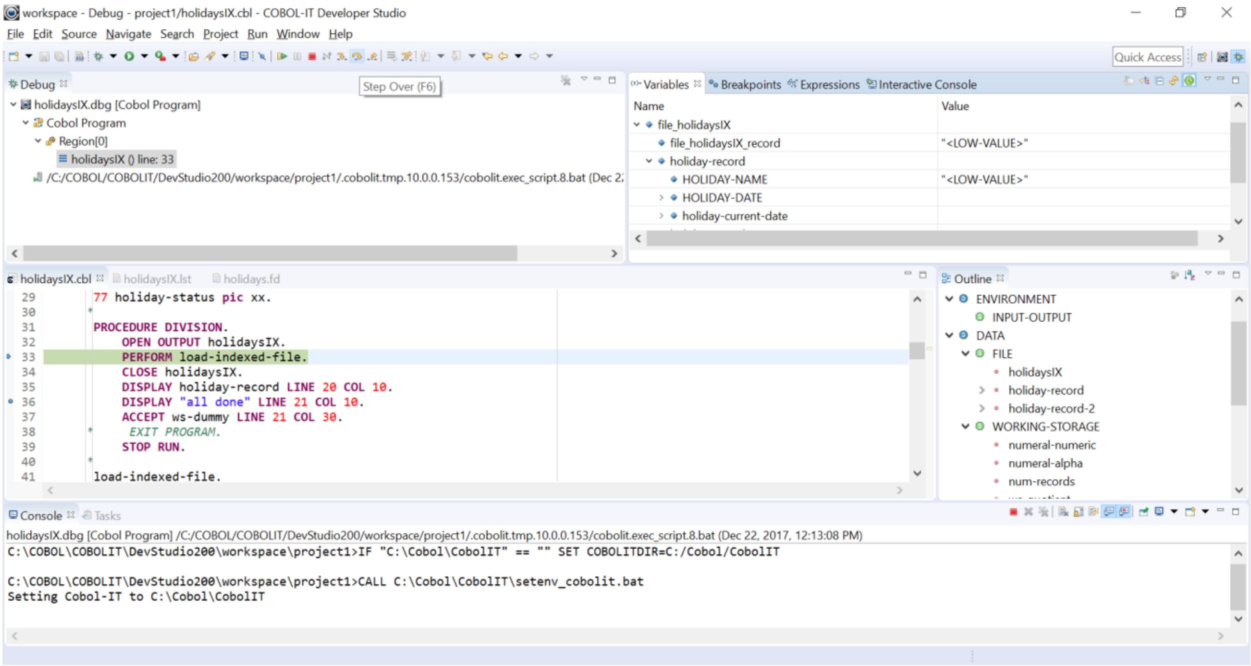
| Name | Value |
|---|---|
| file_holidaysIX | |
| file_holidaysIX_record | "<LOW-VALUE>" |
| holiday-record | |
| HOLIDAY-NAME | "<LOW-VALUE>" |
| HOLIDAY-DATE | |
| holiday-current-date | |

```
29        77 holiday-status pic xx.
30       *
31        PROCEDURE DIVISION.
32            OPEN OUTPUT holidaysIX.
33            PERFORM load-indexed-file.
34            CLOSE holidaysIX.
35            DISPLAY holiday-record LINE 20 COL 10.
36            DISPLAY "all done" LINE 21 COL 10.
37            ACCEPT ws-dummy LINE 21 COL 30.
38       *     EXIT PROGRAM.
39            STOP RUN.
40       *
41        load-indexed-file.
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Dec 22, 2017, 12:13:08 PM)
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

### *Step Over*

With the cursor positioned on a PERFORM or CALL statement, use the Step Over/Fn+F6 function to cause all of the code in the target of the PERFORM/CALL to be executed, and then advance to the next line of code. In this case, the cursor will then be positioned on line:
35: CLOSE holidaysIX.



| Name | Value |
|---|---|
| file_holidaysIX | |
| file_holidaysIX_record | "<LOW-VALUE>" |
| holiday-record | |
| HOLIDAY-NAME | "<LOW-VALUE>" |
| HOLIDAY-DATE | |
| holiday-current-date | |

```
29        77 holiday-status pic xx.
30       *
31        PROCEDURE DIVISION.
32            OPEN OUTPUT holidaysIX.
33            PERFORM load-indexed-file.
34            CLOSE holidaysIX.
35            DISPLAY holiday-record LINE 20 COL 10.
36            DISPLAY "all done" LINE 21 COL 10.
37            ACCEPT ws-dummy LINE 21 COL 30.
38       *     EXIT PROGRAM.
39            STOP RUN.
40       *
41        load-indexed-file.
```

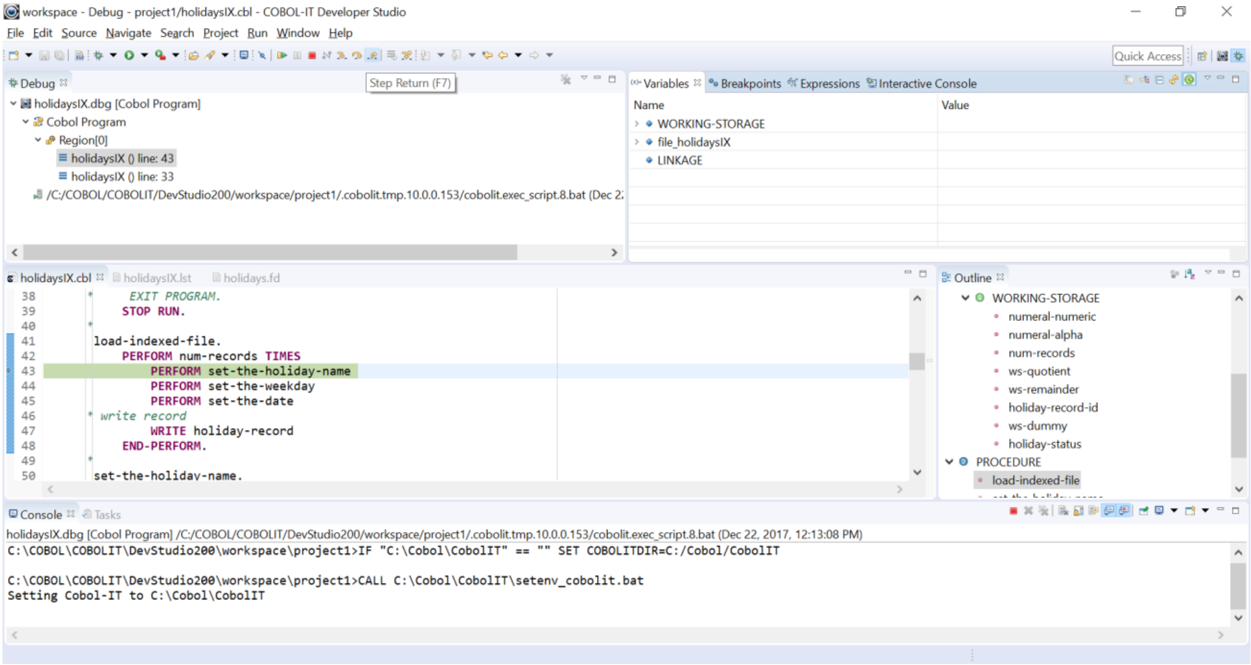
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Dec 22, 2017, 12:13:08 PM)

```
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

### *Step Return*

With the cursor positioned inside the target of a PERFORM statement, on a line of code, use the Step Return/Fn+F7 function to execute the rest of the code in the target paragraph, and then advance to the next line of code. In this case, the cursor will then be positioned on line: 35: CLOSE holidaysIX.

This action of the Step Return (F7) command will pause on Format 1/Format 2 ACCEPT Statements, and will pause on breakpoints.

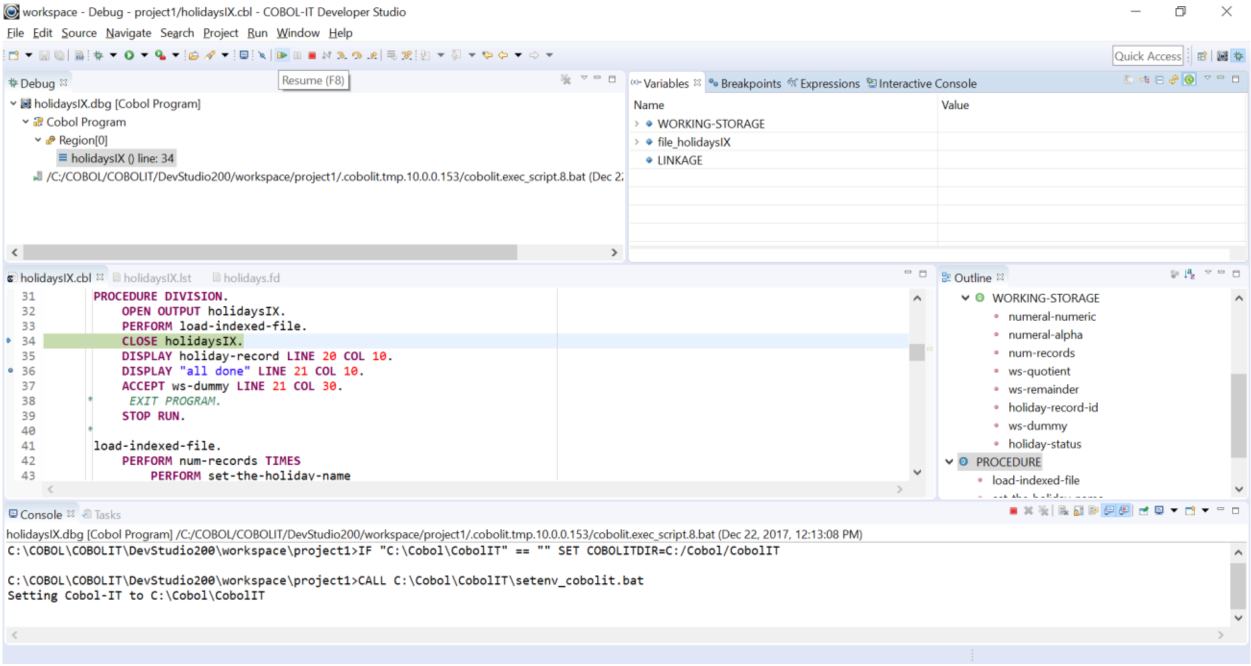


```
      *      EXIT PROGRAM.
           STOP RUN.
      *
       load-indexed-file.
           PERFORM num-records TIMES
               PERFORM set-the-holiday-name
               PERFORM set-the-weekday
               PERFORM set-the-date
      * write record
               WRITE holiday-record
           END-PERFORM.
      *
       set-the-holiday-name.
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Dec 22, 2017, 12:13:08 PM)
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

### *Resume*

The Resume (F8) command causes the program to run normally until it reaches a breakpoint or until the program is terminated. When a breakpoint is reached, the program re-enters a debugging mode, and stepping operations can be resumed. In this case, the cursor will then be positioned at the breakpoint on line:
36: DISPLAY “all done” line 21 col 30.



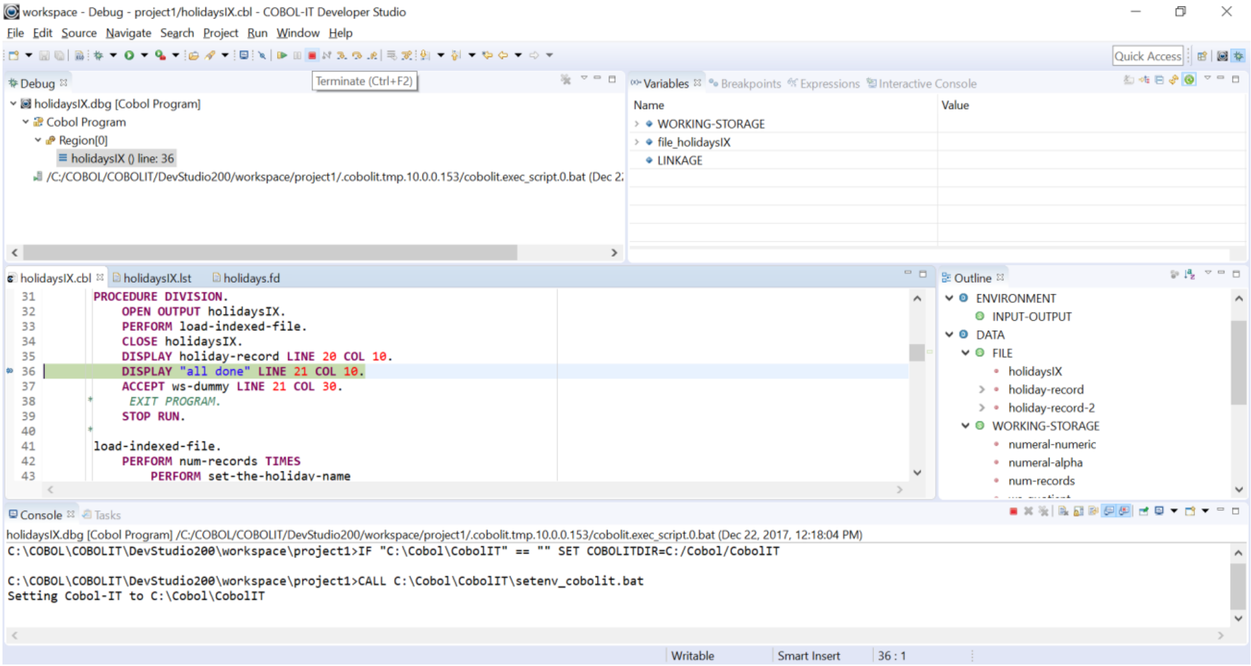
```
       PROCEDURE DIVISION.
           OPEN OUTPUT holidaysIX.
           PERFORM load-indexed-file.
           CLOSE holidaysIX.
           DISPLAY holiday-record LINE 20 COL 10.
           DISPLAY "all done" LINE 21 COL 10.
           ACCEPT ws-dummy LINE 21 COL 30.
      *     EXIT PROGRAM.
           STOP RUN.
      *
       load-indexed-file.
           PERFORM num-records TIMES
               PERFORM set-the-holiday-name
```

holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Dec 22, 2017, 12:13:08 PM)

```
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

## *Terminate*

The Terminate command terminates debugging session, and terminates the program at the current line of execution.



```
       PROCEDURE DIVISION.
           OPEN OUTPUT holidaysIX.
           PERFORM load-indexed-file.
           CLOSE holidaysIX.
           DISPLAY holiday-record LINE 20 COL 10.
           DISPLAY "all done" LINE 21 COL 10.
           ACCEPT ws-dummy LINE 21 COL 30.
      *     EXIT PROGRAM.
           STOP RUN.
      *
       load-indexed-file.
           PERFORM num-records TIMES
               PERFORM set-the-holiday-name
```

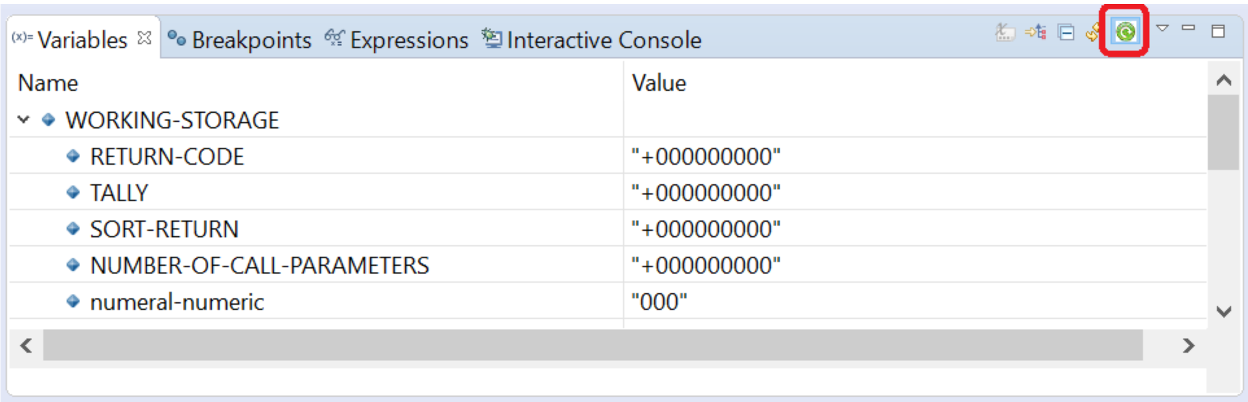
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.0.bat (Dec 22, 2017, 12:18:04 PM)

```
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

## The Variables View

### *Refreshing the Variable View*

By default, the COBOL-IT Debugger Perspective is configured to Auto-Refresh the Variable view.

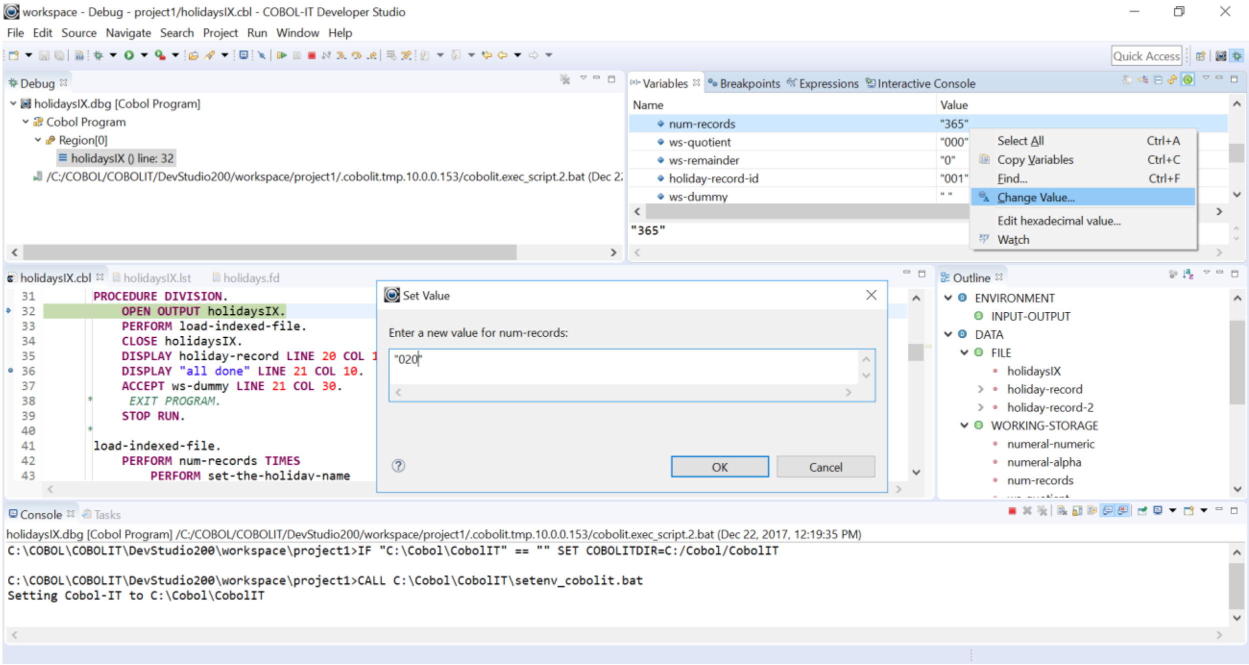


| Name | Value |
|---|---|
| WORKING-STORAGE | |
| RETURN-CODE | "+000000000" |
| TALLY | "+000000000" |
| SORT-RETURN | "+000000000" |
| NUMBER-OF-CALL-PARAMETERS | "+000000000" |
| numeral-numeric | "000" |

If you have a very large Working-Storage Section, this may not be desirable.  Under these conditions, it could be preferable for the user to select the Manual Refresh on the Variables View toolbar, and use the Expressions View for auto-refresh of selected variables.

### *Change the value of a variable*

In the Variable View, you can change the value of a variable by selecting the variable in the Variable View, right-clicking, and selecting Change Value…   In the Change Value dialog screen,  type over the existing value, and the Variable View will be updated with the new value. Debugging can proceed with the new variable value.



| Name | Value |
|---|---|
| num-records | "365" |
| ws-quotient | "000" |
| ws-remainder | "0" |
| holiday-record-id | "001" |
| ws-dummy | " " |

```
"365"
```

```
       PROCEDURE DIVISION.
           OPEN OUTPUT holidaysIX.
           PERFORM load-indexed-file.
           CLOSE holidaysIX.
           DISPLAY holiday-record LINE 20 COL 1
           DISPLAY "all done" LINE 21 COL 10.
           ACCEPT ws-dummy LINE 21 COL 30.
      *     EXIT PROGRAM.
           STOP RUN.
      *
       load-indexed-file.
           PERFORM num-records TIMES
               PERFORM set-the-holiday-name
```

Enter a new value for num-records:

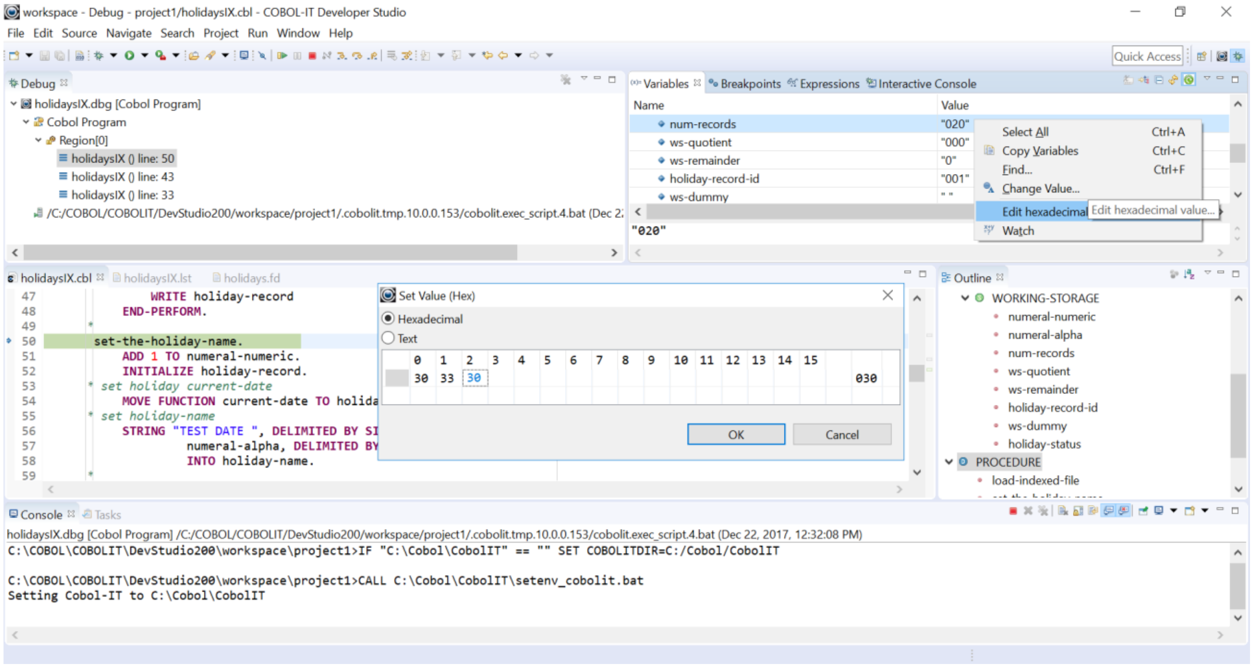
"020"

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.2.bat (Dec 22, 2017, 12:19:35 PM)
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

Verify that the change has been made, and continue with debugging.

### *Change the hexadecimal value of a variable*

The COBOL value dialog screen allows the user to set the value of a variable in either Hexadecimal or Text. In the Variable View, you can change the COBOL value of a variable by selecting the variable in the Variable View, right-clicking, and selecting Edit hexadecimal value… The Set Value (hex) dialog screen allows you to enter a new value for the selected variable. Type over the existing value, and the Variable View will be updated with the new value. Debugging can proceed with the new variable value.



| Name | Value |
|---|---|
| num-records | "020" |
| ws-quotient | "000" |
| ws-remainder | "0" |
| holiday-record-id | "001" |
| ws-dummy | " " |

"020"

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 30 | 33 | 30 | | | | | | | | | | | | | | 030 |

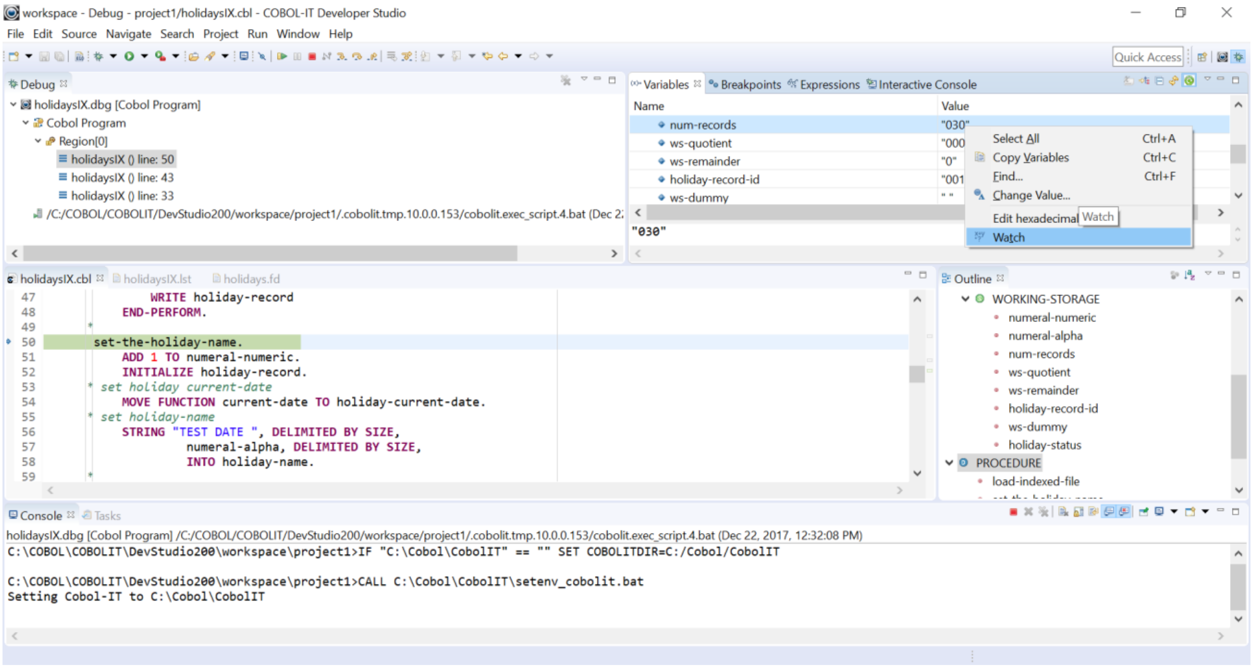
```
            WRITE holiday-record
        END-PERFORM.
      *
    set-the-holiday-name.
        ADD 1 TO numeral-numeric.
        INITIALIZE holiday-record.
      * set holiday current-date
        MOVE FUNCTION current-date TO holida
      * set holiday-name
        STRING "TEST DATE ", DELIMITED BY SI
                 numeral-alpha, DELIMITED BY
                 INTO holiday-name.
      *
```

holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.4.bat (Dec 22, 2017, 12:32:08 PM)

```
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

## *Create a Watch for a variable*

You can add a Variable to the Expressions View  by selecting the variable in the Variable View, right-clicking, and selecting Watch…. The Variable and its current value are transferred into the  Expressions View Window.   De-select "Auto-refresh" when using Expressions for best performance.



```
holidaysIX.dbg [Cobol Program]
  Cobol Program
    Region[0]
      holidaysIX () line: 50
      holidaysIX () line: 43
      holidaysIX () line: 33
  /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.4.bat (Dec 2:
```

| Name | Value |
|---|---|
| num-records | "030" |
| ws-quotient | "000 |
| ws-remainder | "0" |
| holiday-record-id | "001 |
| ws-dummy | " " |

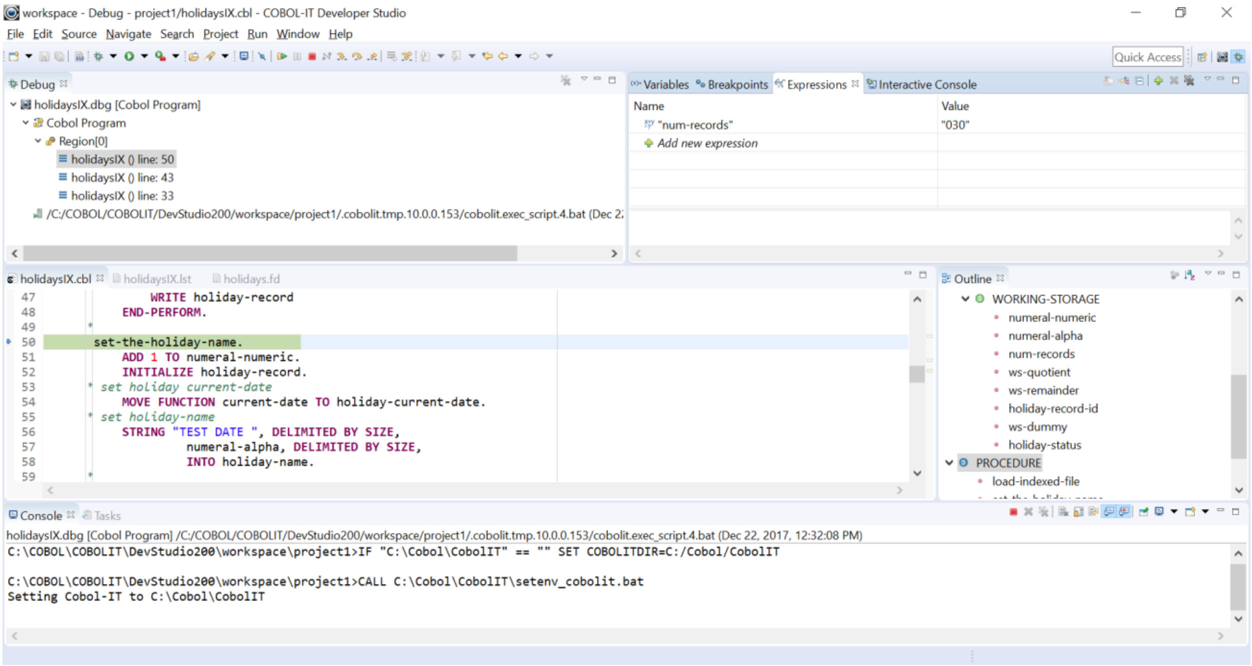
"030"

```
               WRITE holiday-record
           END-PERFORM.
      *
       set-the-holiday-name.
           ADD 1 TO numeral-numeric.
           INITIALIZE holiday-record.
      * set holiday current-date
           MOVE FUNCTION current-date TO holiday-current-date.
      * set holiday-name
           STRING "TEST DATE ", DELIMITED BY SIZE,
                    numeral-alpha, DELIMITED BY SIZE,
                    INTO holiday-name.
      *
```

```
WORKING-STORAGE
  numeral-numeric
  numeral-alpha
  num-records
  ws-quotient
  ws-remainder
  holiday-record-id
  ws-dummy
  holiday-status
PROCEDURE
  load-indexed-file
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.4.bat (Dec 22, 2017, 12:32:08 PM)
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

Open the Expressions View to limit the number of variables you are following.


workspace - Debug - project1/holidaysIX.cbl - COBOL-IT Developer Studio
File Edit Source Navigate Search Project Run Window Help
Quick Access
Debug
Variables
Breakpoints
Expressions
Interactive Console
holidaysIX.cbl
holidaysIX.lst
holidays.fd
Outline
Console
Tasks

| Name | Value |
|---|---|
| "num-records" | "030" |
| *Add new expression* | |

```
holidaysIX.dbg [Cobol Program]
  Cobol Program
    Region[0]
      holidaysIX () line: 50
      holidaysIX () line: 43
      holidaysIX () line: 33
  /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.4.bat (Dec 2:
```

```
               WRITE holiday-record
           END-PERFORM.
      *
       set-the-holiday-name.
           ADD 1 TO numeral-numeric.
           INITIALIZE holiday-record.
      * set holiday current-date
           MOVE FUNCTION current-date TO holiday-current-date.
      * set holiday-name
           STRING "TEST DATE ", DELIMITED BY SIZE,
                    numeral-alpha, DELIMITED BY SIZE,
                    INTO holiday-name.
      *
```

```
WORKING-STORAGE
  numeral-numeric
  numeral-alpha
  num-records
  ws-quotient
  ws-remainder
  holiday-record-id
  ws-dummy
  holiday-status
PROCEDURE
  load-indexed-file
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.4.bat (Dec 22, 2017, 12:32:08 PM)
C:\COBOL\COBOLIT\DevStudio200\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

C:\COBOL\COBOLIT\DevStudio200\workspace\project1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
Setting Cobol-IT to C:\Cobol\CobolIT
```

## Breakpoints

### *Toggle Breakpoints from within the Editor Window*

You can toggle a line breakpoint ( set or unset ) by double-clicking in the left-most column of the Editor Window. This is the column to the left of the line-number columns, and is where the small circle that represents a breakpoint can visually be seen in your source code.

### *Toggle Breakpoints from within the Breakpoints View Window*

When you have created a breakpoint, you will see that it has been added to the Breakpoints View window, and that the enabling checkbox is checked. To remove a breakpoint from the Breakpoints View, select the breakpoint, open the right-click dropdown menu, and select the Remove function.

### *Disable a breakpoint*

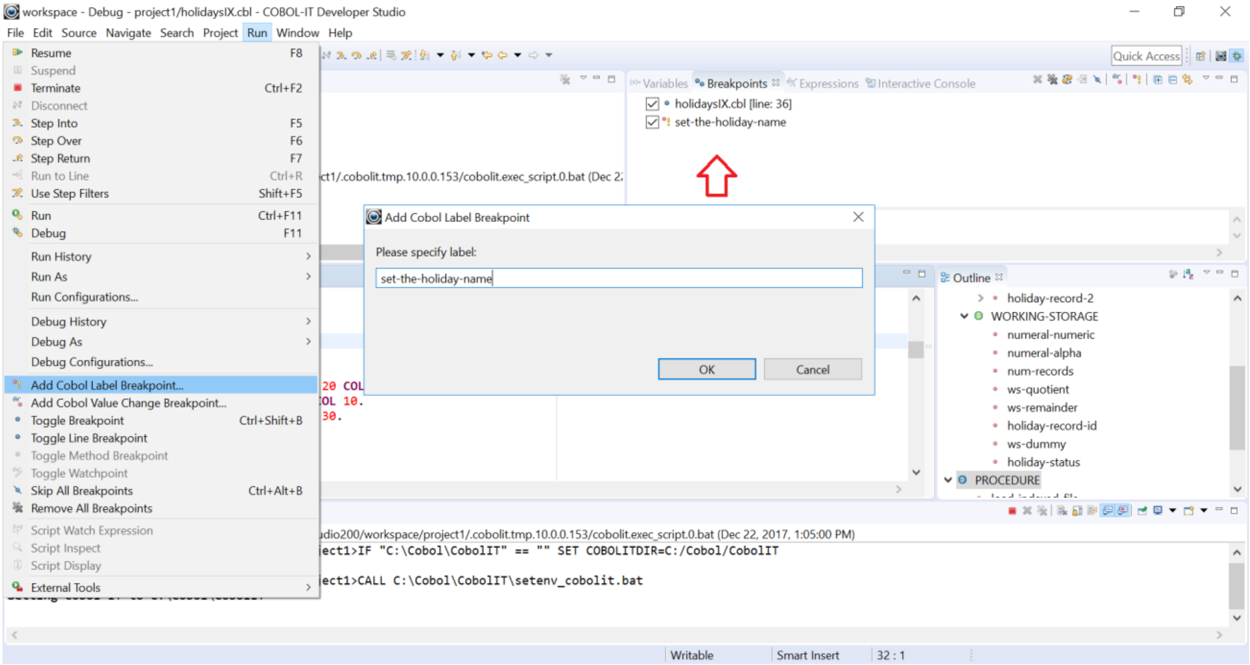
You can Disable a breakpoint by right-clicking on the breakpoint in the COBOL-IT Program View, and selecting the Disable function, or by right-clicking on the breakpoint declaration in the Breakpoints View screen, and selecting the Disable function, or by de-selecting the breakpoint's enabling checkbox in the Breakpoints View screen. Note- Disabling a breakpoint is different than removing a breakpoint in that it can subsequently be enabled. You can enable a disabled breakpoint using the same right-click interfaces used to disable the breakpoint.

### *Skip all breakpoints*

You can disable all breakpoints by clicking on the Skip All Breakpoints pushbutton on the Breakpoints View toolbar.

### *Run>Add COBOL Label Breakpoint…*

Select the Run function on the main menubar, right-click, and select Add COBOL Label Breakpoint… from the dropdown menu. Enter the label name of a paragraph or section and click OK. With a COBOL Label Breakpoint, a breakpoint condition occurs every time that the label is encountered.



☑ holidaysIX.cbl [line: 36]
☑ set-the-holiday-name

Add Cobol Label Breakpoint

Please specify label: set-the-holiday-name

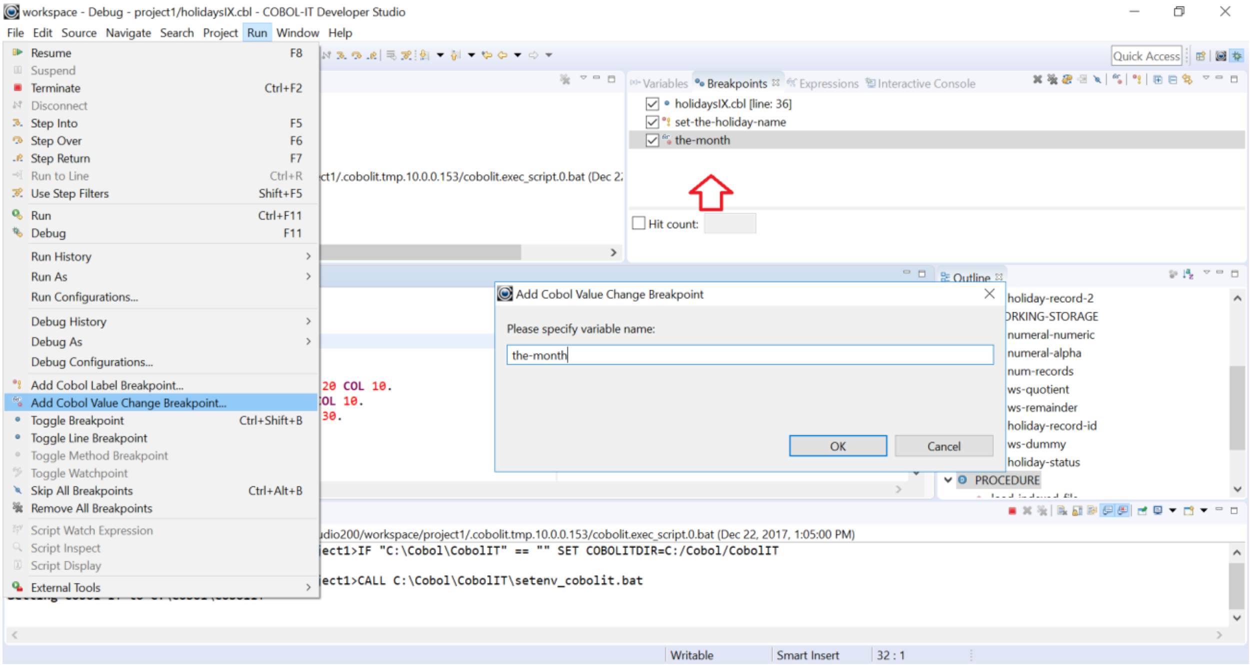
OK Cancel

```
ct1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.0.bat (Dec 2
```

```
dio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.0.bat (Dec 22, 2017, 1:05:00 PM)
ect1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

ect1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
```

## *Run>Add COBOL Value Change Breakpoint*

Select the Run function on the main menubar, right-click, and select Add COBOL Value Change Breakpoint… from the dropdown menu. Enter the variable name, and click OK. With a COBOL Value Change Breakpoint, a breakpoint condition occurs every time that the value of this variable changes. The COBOL Value Change Breakpoint can be used with a variable being watched in the Expressions View.



☑ holidaysIX.cbl [line: 36]
☑ set-the-holiday-name
☑ the-month

□ Hit count:

Add Cobol Value Change Breakpoint

Please specify variable name:

the-month

OK Cancel

holiday-record-2
ORKING-STORAGE
numeral-numeric
numeral-alpha
num-records
ws-quotient
ws-remainder
holiday-record-id
ws-dummy
holiday-status
PROCEDURE

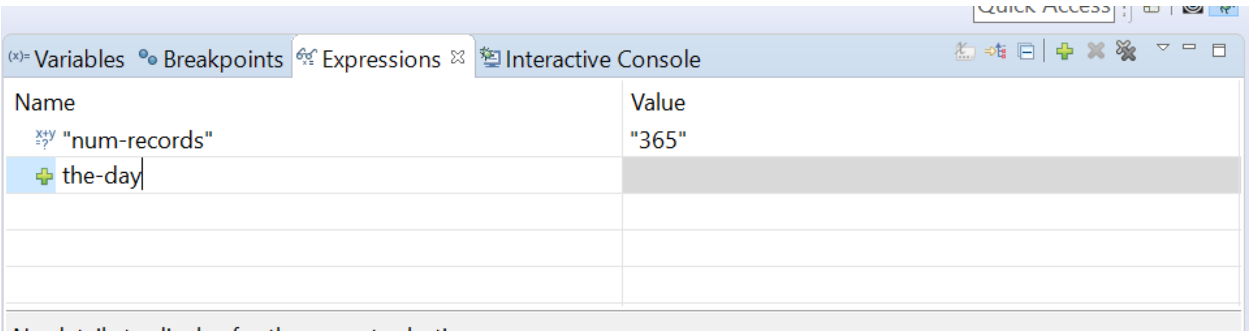
```
20 COL 10.
OL 10.
30.
```

```
udio200/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.0.bat (Dec 22, 2017, 1:05:00 PM)
ect1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT

ect1>CALL C:\Cobol\CobolIT\setenv_cobolit.bat
```

## Expressions

### *Add a new expression*

To add a new expression, you can click on the “Add new expression” button on the Expressions toolbar, or you can click on the in-line “Add new expression” function. Clicking on the inline “Add new expression” function allows the user to type the expression directly into the “Name” column.



| Name | Value |
|---|---|
| "num-records" | "365" |
| the-day | |

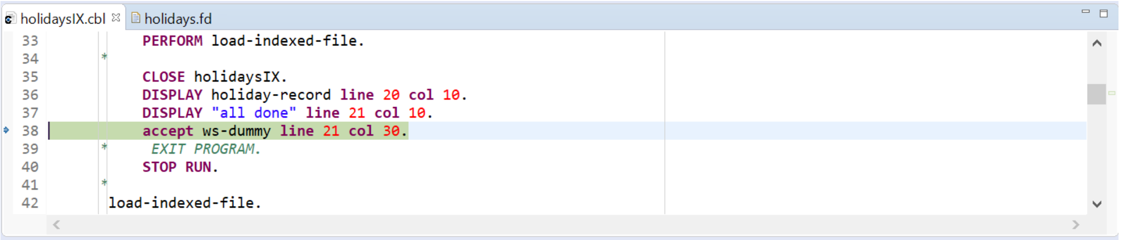
## Code Editor Functions

### *Show Line Numbers*

You can Show or Hide Line Numbers by right-clicking in the left-most column of the Editor Window, and checking or unchecking the Show Line Numbers function.

### *Transferring control to console on ACCEPT*

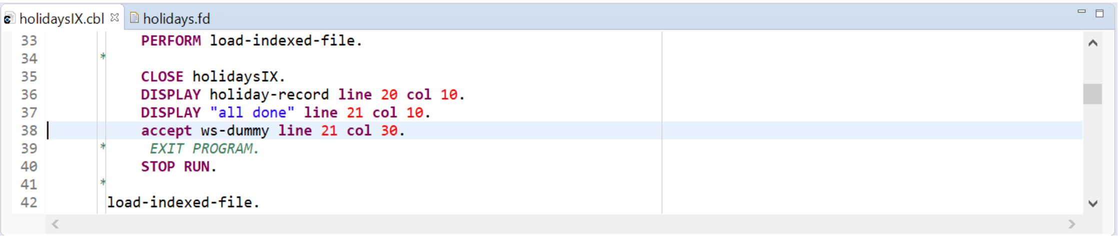
The behavior of the COBOL-IT Debugger when positioned on an ACCEPT statement is worth noting. When stepping through code, the current line is marked by an arrow in the left-most column of the current line. The source code on the line is colorized (green) before the code is executed.



```
    PERFORM load-indexed-file.
*
    CLOSE holidaysIX.
    DISPLAY holiday-record line 20 col 10.
    DISPLAY "all done" line 21 col 10.
    accept ws-dummy line 21 col 30.
*    EXIT PROGRAM.
    STOP RUN.
*
load-indexed-file.
```

Normally, the single-step process will move this arrow, and this colorized source line by line through the code. However, in the case of the ACCEPT statement, the debugger must pause, and wait for input from the console.

In the case above, press F5 to single-step. This executes the ACCEPT statement, but does not move the line. The debugger signals the user that it is paused and waiting for input by changing the colorization of the current line. See below:



```
    PERFORM load-indexed-file.
*
    CLOSE holidaysIX.
    DISPLAY holiday-record line 20 col 10.
    DISPLAY "all done" line 21 col 10.
    accept ws-dummy line 21 col 30.
*    EXIT PROGRAM.
    STOP RUN.
*
load-indexed-file.
```

At this point, the user must raise the console, and perform the operation that terminates the accept. In this case, hit the [Enter] key.

# COBOL-IT Debugger Reference

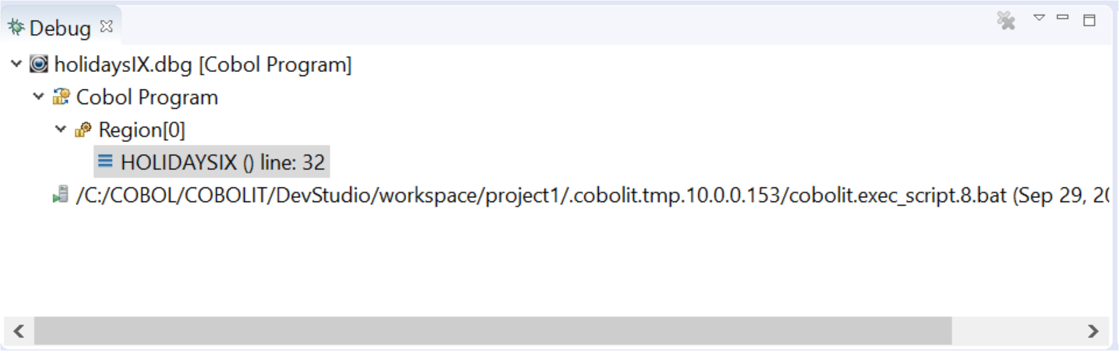
# The Views in the COBOL-IT Debugger Perspective

The functionality of the COBOL-IT Debugger Perspective is presented through the View interfaces. A quick overview of the main Views in the COBOL-IT Debugger Perspective is instructive:

| Debugger Perspective View | Functionality |
|---|---|
| The Debug View | A view of the stack executing in the Debugger |
| The COBOL-IT Program View | A COBOL-IT Code Editor which is animated during debugging |
| The Variable View | Working-Storage, Linkage Section, File Section can be expanded to see values |
| The Breakpoints View | Lists status of all breakpoints |
| The Expressions View | Lists all Watches that have been set |
| The Outline View | An Outline of the active source file |
| The Console View | Consoles include the COBOL-IT Compiler, runtime, and debugger consoles. |
| The Tasks View | Provides an interface for tracking tasks |
| The Problems View | Provides a clickable interface for locating compiler errors in source code. |

# The Debug View

The Debug View shows a stack view of the execution of  your program.  In the graphic below, you can see the the program is currently executing at line 32 in the the program HOLIDAYSIX.



holidaysIX.dbg [Cobol Program]
Cobol Program
Region[0]
HOLIDAYSIX () line: 32
/C:/COBOL/COBOLIT/DevStudio/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Sep 29, 2(

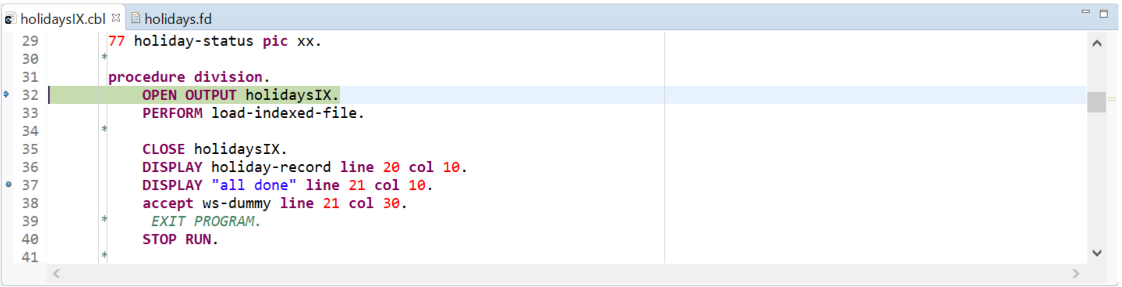
### *The Debug view toolbar*

The functions that are presented on the Debug View toolbar help manage the visual display of the call stack in the View, and the commands used to navigate inside a debugging session.

| | | |
|---|---|---|
| | Remove all terminated launches | Clears terminated launches from the Debug View |
| | Connect to a process | Initiates a Remote Debugging session. |
| ▶ | Resume (F8) | Resumes the execution of the program. |
| ⏸ | Suspend | Suspends execution of the program, and re-enters the debugger. |
| 🟥 | Terminate (Ctrl+F2) | Terminates the currently running process. |
| | Disconnect | Terminates a Remote Debugging session. |
| | Step Into (F5) | The Step Into function single steps through lines of code. The Step Into function enters paragraphs/subprograms that are the targets of PERFORM / CALL statements. |
| | Step Over (F6) | The Step Over Into function causes paragraphs/subprograms that are the targets of PERFORM / CALL statements to be executed, and then Steps to the next line of code. |
| | Step Return (F7) | The Step Return function causes code in a paragraph/section/subprogram to be executed, and then Steps to the next line of code after the parent PERFORM/CALL statement. |
| | Drop to Frame | Not Supported in the COBOL-IT Debugger Perspective |
| i⇨ | Instruction Stepping Mode | Not Supported in the COBOL-IT Debugger Perspective |
| | Use Step Filters | Not Supported in the COBOL-IT Debugger Perspective |
| ▽ | View Menu | Drops down a view of Menu options |

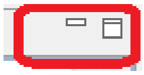
# The COBOL-IT Program View

The COBOL-IT Program View is a COBOL Code Editor that is responsive to the commands described in the Debug View. In the graphic below, note the arrow ➡ positioned on line 32. This marks the line of code that the debugger will execute next. Note also the circle ● positioned on line 37. This marks a breakpoint, that is set within the source code.



```
29        77 holiday-status pic xx.
30       *
31        procedure division.
32            OPEN OUTPUT holidaysIX.
33            PERFORM load-indexed-file.
34       *
35            CLOSE holidaysIX.
36            DISPLAY holiday-record line 20 col 10.
37            DISPLAY "all done" line 21 col 10.
38            accept ws-dummy line 21 col 30.
39       *     EXIT PROGRAM.
40            STOP RUN.
41       *
```

### *The COBOL-IT Program View Toolbar*

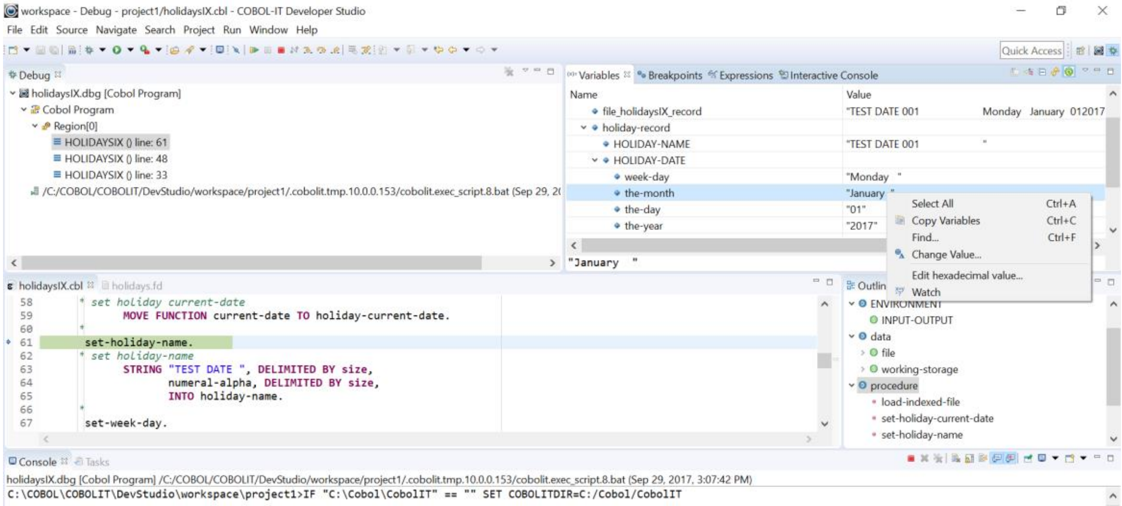


The functions that are presented on the COBOL-IT Program View toolbar allow the user to either Minimize or Maximize the Code Editor Window.

# The Variable View

The Variable View provides a graphical user interface in which the variables/values of the File Section, Working-Storage Section, and Linkage Section are displayed. The Right-Click drop-down menu from a selected Variable in the Variable View provides:

| | |
|---|---|
| Select All Ctrl+A | Select all variables and copy to clipboard |
| Copy Variables | Select one or more variables and copy to clipboard |
| Find... | Search for a variable name in the Variable View |
| Change Value... | Modify the value (Ascii) of a variable |
| Edit hexadecimal value… | Modify the value (Hexadecimal) of a variable |
| Watch | Copy a variable into the Expressions View |



| Name | Value |
|---|---|
| file_holidaysIX_record | "TEST DATE 001 Monday January 012017 |
| holiday-record | |
| HOLIDAY-NAME | "TEST DATE 001 " |
| HOLIDAY-DATE | |
| week-day | "Monday " |
| the-month | "January " |
| the-day | "01" |
| the-year | "2017" |

```
"January  "
```

```
      * set holiday current-date
            MOVE FUNCTION current-date TO holiday-current-date.
      *
       set-holiday-name.
      * set holiday-name
            STRING "TEST DATE ", DELIMITED BY size,
                   numeral-alpha, DELIMITED BY size,
                   INTO holiday-name.
      *
       set-week-day.
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio/workspace/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.8.bat (Sep 29, 2017, 3:07:42 PM)
C:\COBOL\COBOLIT\DevStudio\workspace\project1>IF "C:\Cobol\CobolIT" == "" SET COBOLITDIR=C:/Cobol/CobolIT
```

### *The Variable View Toolbar*

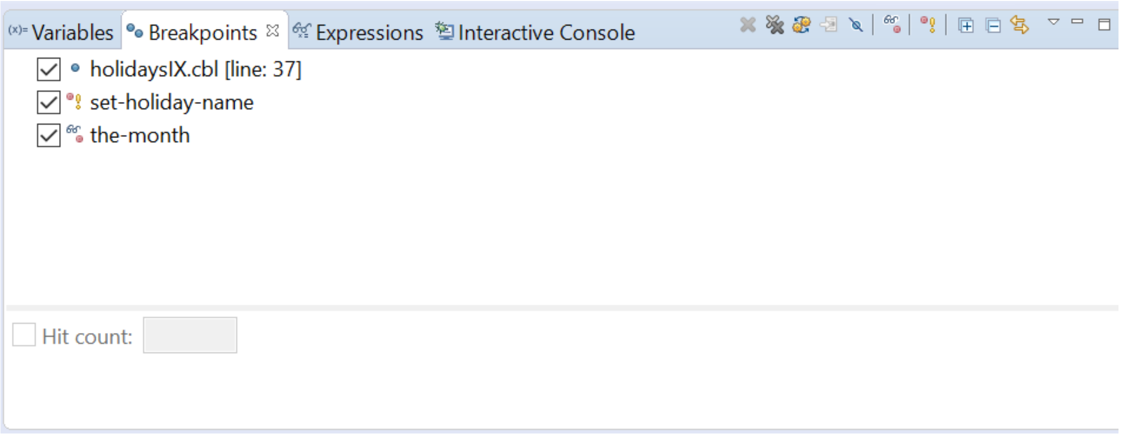
The functions that are presented on the Variable View toolbar allow the user to expand/collapse groups of variables. Options exist for Auto-Refresh and Manual Refresh of variable values.

| | | |
|---|---|---|
| | Show Type Names | Not Supported in the COBOL-IT Debugger Perspective |
| | Show Logical Structure | COBOL data items are all presented in their logical structures, and can be expanded and collapsed. |
| | Collapse All (Ctrl+Shift+Numpad-Divide) | Collapses all structures back to their parent Sections ( Working-Storage, File, Linkage ) |
| | Refresh | Manual Refresh function. In programs with large numbers of variables, avoids degradation |

| | | |
|---|---|---|
| | | of performance associated with Auto-Refresh. |
| ⟳ | Auto Refresh | Auto-Refresh function. Working Storage, File Section, and Linkage Section are auto-refreshed in real time, while debugging the program. |
| | Add Global Variables | Not Supported in the Debugger Perspective |
| ✖ | Remove Selected Global Variables | Not Supported in the Debugger Perspective |
| ✖ | Remove All Global Variables | Not Supported in the Debugger Perspective |
| ▽ | View Menu | Drops down a view of Menu options |

# The Breakpoints View

The Breakpoints View is a single interface for managing all of the breakpoints in all of the programs that you have open in the Debugger Perspective. Breakpoints can be set at line numbers, on labels, or on variable value change.

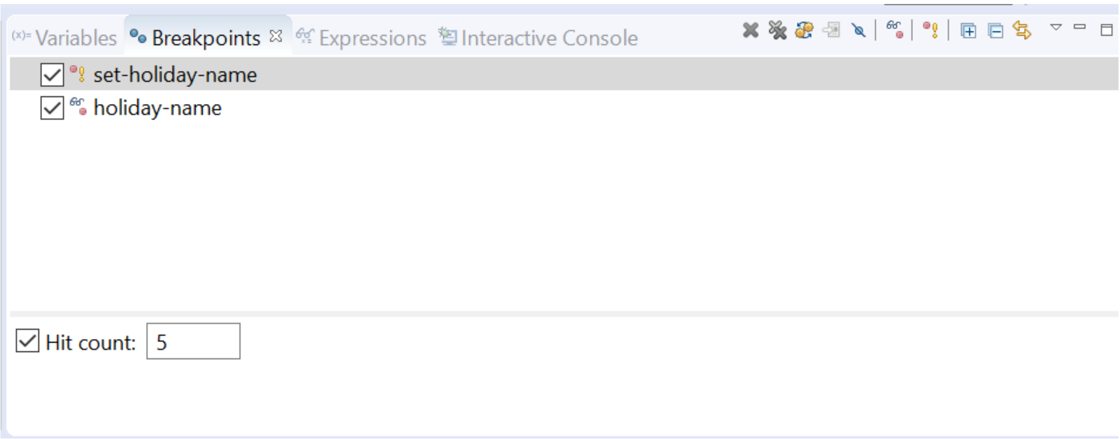


☑ holidaysIX.cbl [line: 37]
☑ set-holiday-name
☑ the-month

□ Hit count:

## *Hit Count*

When you create a breakpoint, you can assign a Hit Count for it. Hit Count is a number. If you set the hit count for a breakpoint to 5, for example, then the breakpoint will occur after the condition has been hit 5 times.



☑ set-holiday-name
☑ holiday-name

☑ Hit count: 5

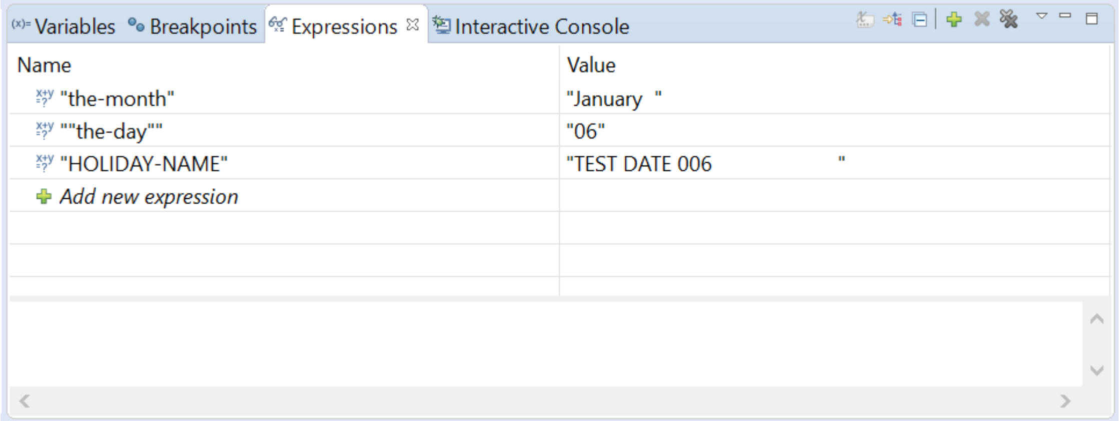
## *The Breakpoints View Toolbar*

The functions that are presented on the Breakpoints View toolbar allow the user to manage breakpoints. Selected breakpoints can be removed, all breakpoints can be removed, .

| | | |
|---|---|---|
| ✖ | Remove Selected Breakpoints | Removes breakpoints that have been selected by clicking in the checkbox in the left-most column. |
| ✖ | Remove All Breakpoints | Removes all breakpoints. |
| ⚙ | Show Breakpoints Supported by Selected Target | Shows all breakpoints set in the COBOL programs. |
| ⇨ | Go To File for Breakpoints | Transfer focus to file at Breakpoint |
| ⊘ | Skip All Breakpoints | Disable ( but do not remove ) all breakpoints. |
| ⊞ | Expand All | Not Supported in the COBOL-IT Debugger Perspective |
| ⊟ | Collapse All | Not Supported in the COBOL-IT Debugger Perspective |
| ⇆ | Link With Debug View | Not Supported in the COBOL-IT Debugger Perspective |
| ▽ | View Menu | Drops down a view of Menu options |

# The Expressions View

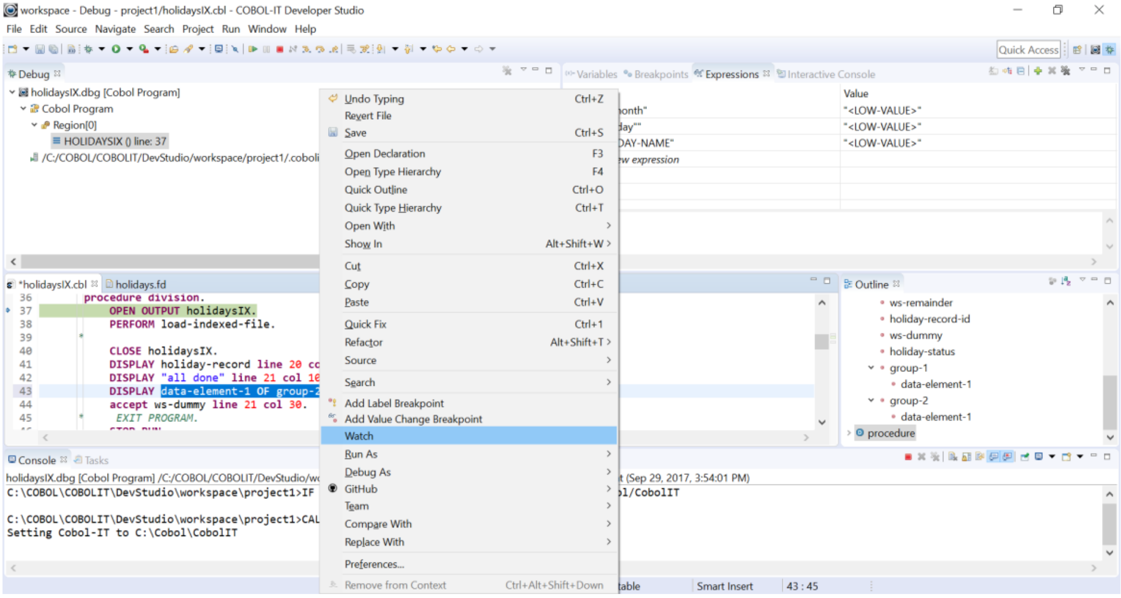
The Expressions View is an interface for managing a select number of variables. If you wish to have an auto-update capability of just a small number of variables, and the performance penalty of running the debugger with Auto-Refresh on for all variables is too high, you can use the Expressions View to minimize the number of variables on Auto-Refresh.



| Name | Value |
|---|---|
| "the-month" | "January " |
| ""the-day"" | "06" |
| "HOLIDAY-NAME" | "TEST DATE 006 " |
| *Add new expression* | |

## Qualified names in the Expression View

The debugger supports qualified names, such as data-element-1 in group-2 in the Expression View.  Navigate to a qualified description of othe data item in the source code, right-click and select Watch from the dropdown menu.

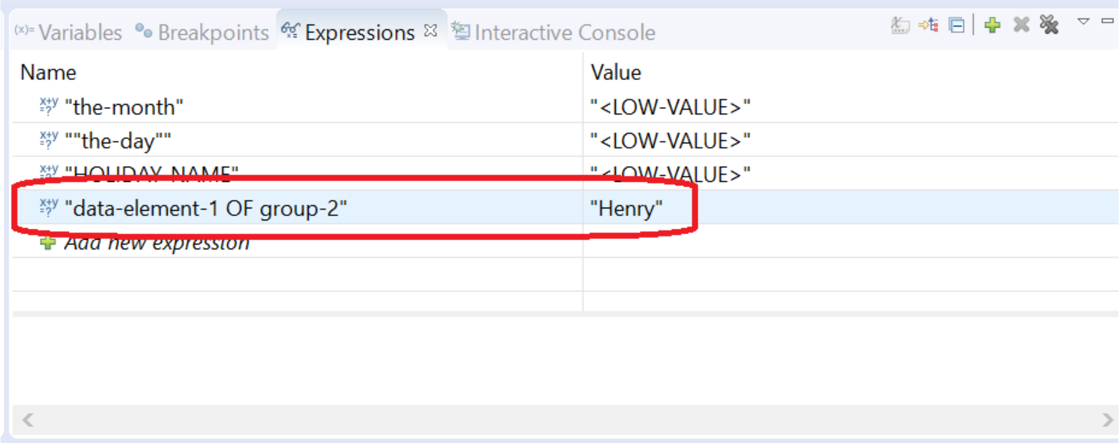


| | Value |
|---|---|
| onth" | "<LOW-VALUE>" |
| day"" | "<LOW-VALUE>" |
| DAY-NAME" | "<LOW-VALUE>" |
| w expression | |

```
       procedure division.
           OPEN OUTPUT holidaysIX.
           PERFORM load-indexed-file.
      *
           CLOSE holidaysIX.
           DISPLAY holiday-record line 20 cc
           DISPLAY "all done" line 21 col 1e
           DISPLAY data-element-1 OF group-2
           accept ws-dummy line 21 col 30.
      *     EXIT PROGRAM.
```

```
holidaysIX.dbg [Cobol Program] /C:/COBOL/COBOLIT/DevStudio/wc                t (Sep 29, 2017, 3:54:01 PM)
C:\COBOL\COBOLIT\DevStudio\workspace\project1>IF                             l/CobolIT

C:\COBOL\COBOLIT\DevStudio\workspace\project1>CAL
Setting Cobol-IT to C:\Cobol\CobolIT
```

The qualified data item is copied to the Expression View.



| Name | Value |
|---|---|
| "the-month" | "<LOW-VALUE>" |
| ""the-day"" | "<LOW-VALUE>" |
| "HOLIDAY-NAME" | "<LOW-VALUE>" |
| "data-element-1 OF group-2" | "Henry" |

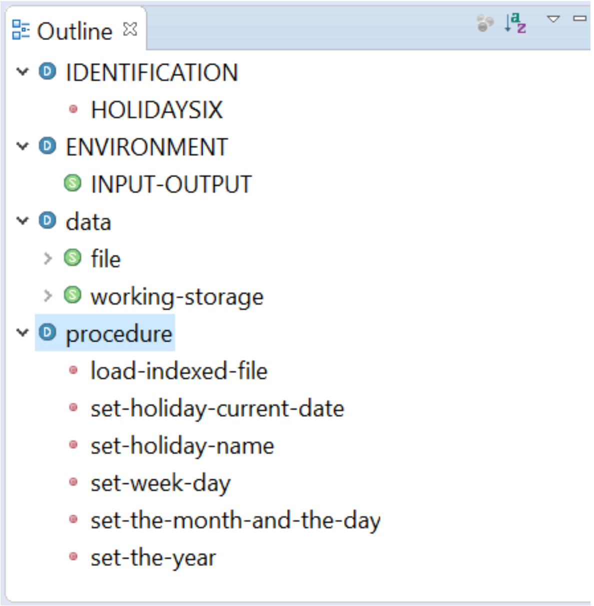
### *The Expressions View Toolbar*

The functions that are presented on the Expressions View toolbar allow the user to manage Expressions. Expressions can be created, collapsed, expanded, and removed.

| | | |
|---|---|---|
| | Show Type Names | Not Supported in the COBOL-IT Debugger Perspective |
| | Show Logical Structure | COBOL data items are all presented in their logical structures, and can be expanded and collapsed. |
| ⊟ | Collapse All (Ctrl+Shift+Numpad-Divide) | Collapses all structures . |
| ✚ | Create a new Watch Expression | Opens dialog window in which a new Watch Expression can be described. Note that Watch Expressions can also be added from within the Variable View, by right-clicking on a variable, and selecting “Watch” from the dropdown menu. |
| ✖ | Remove Selected Expressions | Removes expressions that have been selected by clicking on them. |
| ✖ | Remove All Expressions | Removes all expressions listed in the Expressions View. |
| ▽ | View Menu | Drops down a view of Menu options |

# The Outline View

The Outline View is an interface for viewing the source file in Outline form. The Outline View is linked to the COBOL-IT Program View, as clicking on any of the entries in the Outline View highlights the corresponding line of code in the COBOL source file.



Outline

- IDENTIFICATION
  - HOLIDAYSIX
- ENVIRONMENT
  - INPUT-OUTPUT
- data
  - file
  - working-storage
- procedure
  - load-indexed-file
  - set-holiday-current-date
  - set-holiday-name
  - set-week-day
  - set-the-month-and-the-day
  - set-the-year

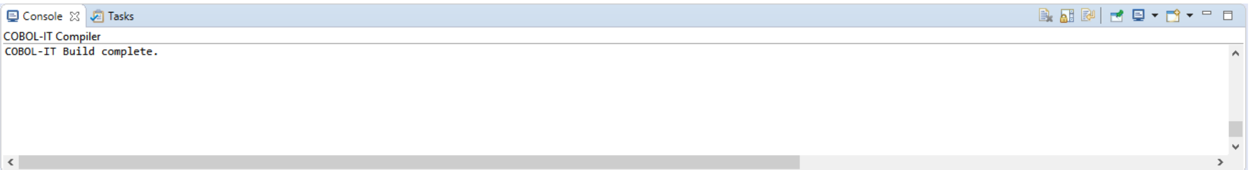
***The Outline View Toolbar***

The Outline View Toolbar contains an Alphabetic Sort function.

| | | |
|---|---|---|
| ⚙ | Focus on Active Task | Not Supported in the COBOL-IT Debugger Perspective |
| ↓az | Sort | Sorts the Outline view alphabetically |
| ▽ | View Menu | Drops down a view of Menu options |

# The Console View

The COBOL-IT Compiler Console View allows you to view the output of the compiler command that has been executed.



```
COBOL-IT Compiler
COBOL-IT Build complete.
```

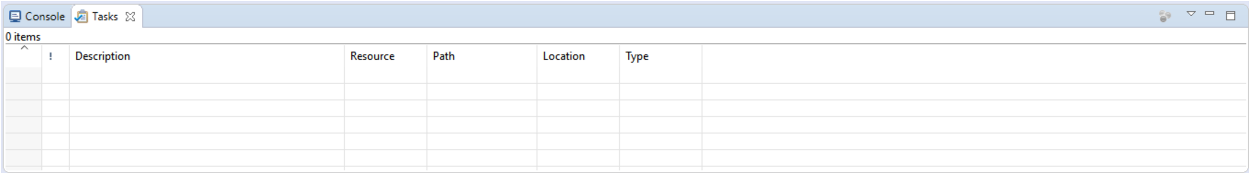
***The Console View Toolbar***

The Console View Toolbar allows you to manage the selection of open consoles, and provides the ability to clear the console.

| | | |
|---|---|---|
| | Clear Console | Clears all text from the open console window. |
| | Scroll Lock | Not Supported in the COBOL-IT Debugger Perspective |
| | Pin Console | Not Supported in the COBOL-IT Debugger Perspective |
| ▾ | Display Selected Console | Allows selection of a console from a dropdown list to be the active console in the output window.   For example, select the COBOL-IT Compiler console to see output from the Compiler on Build operations. |
| ▾ | Open Console | Adds a new console to the Display Selected Console dropdown list of available consoles. |

## The Tasks View

The Tasks View toolbar provides access to the Mylyn Task Manager.  For details on how to use the Task Manager, see Getting Started with the Developer Studio- The Utilities.



0 items

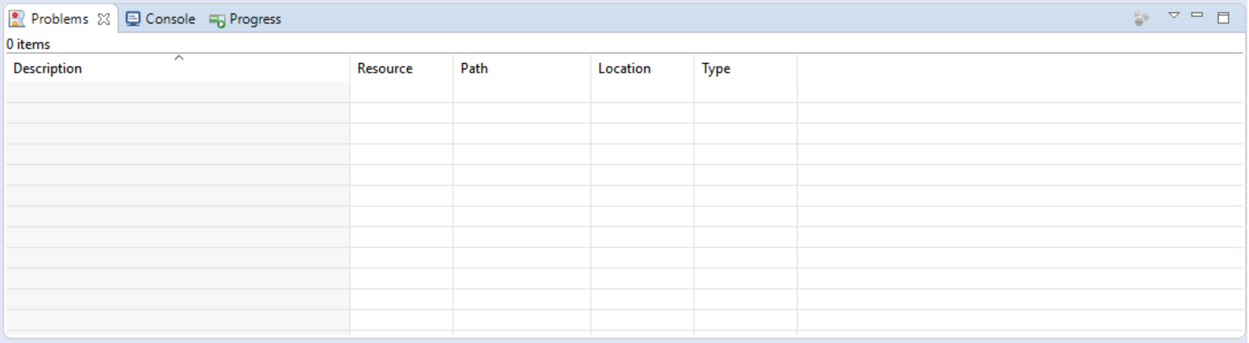
| ! | Description | Resource | Path | Location | Type |
|---|---|---|---|---|---|

***The Tasks View Toolbar***

| | | |
|---|---|---|
| | Focus on Active Task | Places focus in active source file. |
| ▽ | View Menu | Drops down a view of Menu options |

# The Problems View

The Problems View provides a clickable interface for returning from reported compiler errors to their location in the source file.



***The Problems View Toolbar***

| | | |
|---|---|---|
| ● | Focus on Active Task | Places focus in active source file. |
| ▽ | View Menu | Drops down a view of Menu options |

# Attaching the Debugger to a Running Process

***Is it possible to attach the COBOL-IT Debugger to a program, or subprogram that is running, at a selected place in the program or subprogram ?***

C$DEBUG is a library routine which can be called using either the PID of the runtime session, or the value of the environment variable COB_DEBUG_ID. Prior to calling C$DEBUG, the program should acquire the value of the PID / COB_DEBUG_ID.

You may acquire the value of the PID of the runtime session by calling the C$PID library routine, using a PIC 9(n) parameter. The parameter must be numeric, and large enough to hold the value of the Process ID.

For example :

```
77 ws-pid       PIC 9(5).
…..
CALL « C$PID » USING ws-pid.
CALL « C$DEBUG » USING ws-pid.
```

You may also call C$DEBUG USING the value of the runtime environment variable COB_DEBUG_ID. Using the runtime environment variable COB_DEBUG_ID to hold the value of this parameter has an advantage if you prefer to set the value of the parameter yourself. Acquire the value of COB_DEBUG_ID programmatically before calling the C$DEBUG library routine. The parameter must be numeric, and large enough to hold the value of the value of the runtime environment variable COB_DEBUG_ID.

For example :

```
77 ws-did     PIC 9(5).
…..
ACCEPT ws-did FROM ENVIRONMENT « COB_DEBUG_ID ».
CALL «C$DEBUG » USING ws-did.
```

After a call to C$DEBUG is made, the executing program, or subprogram is paused. In this state, the COBOL-IT Debugger may be attached to this runtime process from the COBOL-IT Developer Studio.

## Key concepts

In order to attach to the COBOL-IT Debugger, the program containing the call to C$DEBUG library routine must be compiled with –g.
The COBOL-IT Developer Studio will request the location of the source file associated with the program/subprogram that has been paused by the C$DEBUG command, for purposes of debugging.
The COBOL-IT Developer Studio attaching to the paused runtime session requires a COBOL Project, and requires that some configuration. Recommended settings are :
Window>Preferences>Run/Debug>Perspectives>Open the associated perspective when lauching (Always )

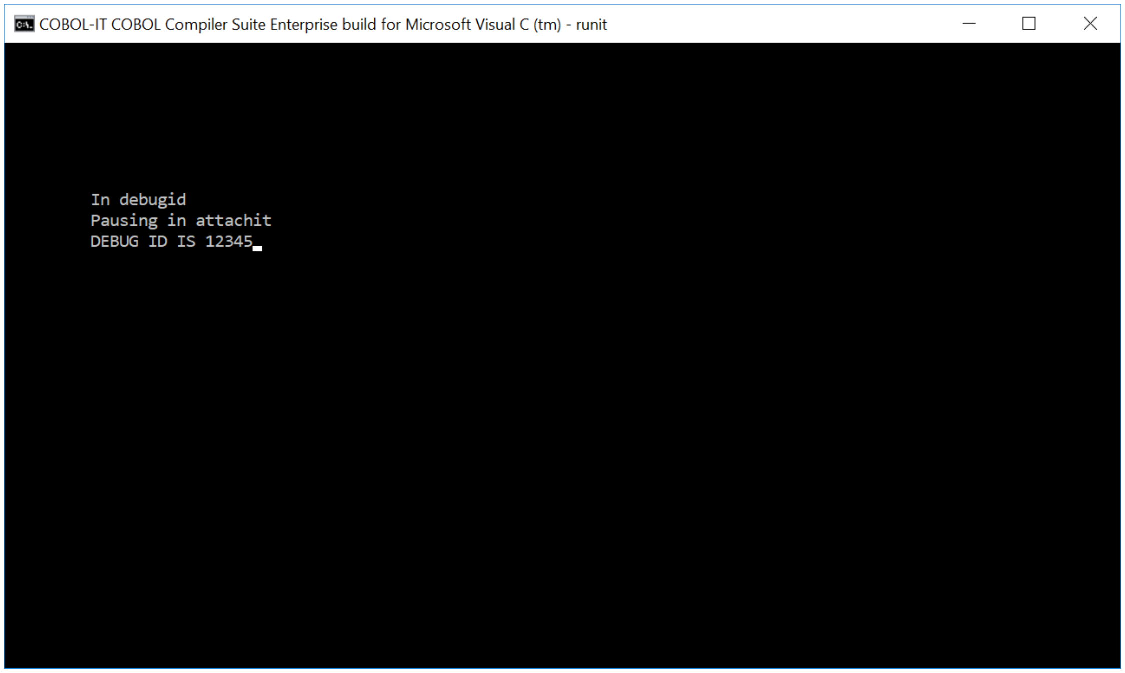
In our example, we have a program, debugid.cbl, which calls a subprogram, subpgm.cbl, which retrieves the PID of the runtime session, and then calls C$DEBUG to pause the runtime session. We will run these programs from a batch file, as follows :

## Launch, and Pause the Runtime using CALL « C$DEBUG »

**runit.bat**

```
SET COB_LIBRARY_PATH=.\object
cobcrun debugid
```

This will return the screen below. Note that in your case, the Process ID will likely be different.

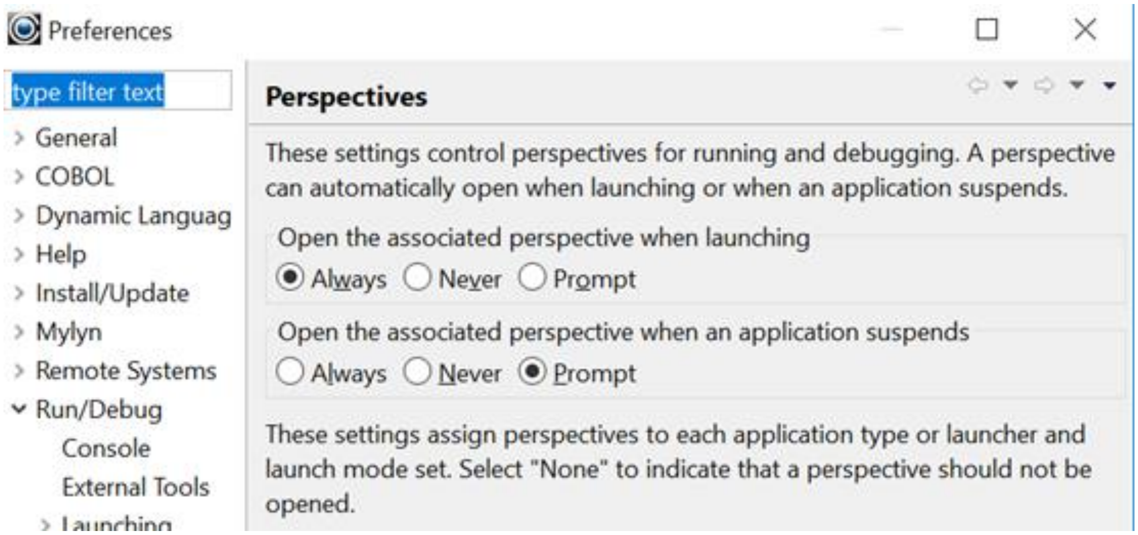


COBOL-IT COBOL Compiler Suite Enterprise build for Microsoft Visual C (tm) - runit

```
In debugid
Pausing in attachit
DEBUG ID IS 12345_
```

## Attach the Debugger from the Developer Studio

The Developer Studio must be configured to enter the Debugger Perspective when launching the debugger.  In Window>Preferences>Run/Debug>Perspectives, set “Open the associated perspective when launching” to “Always”.



Perspectives

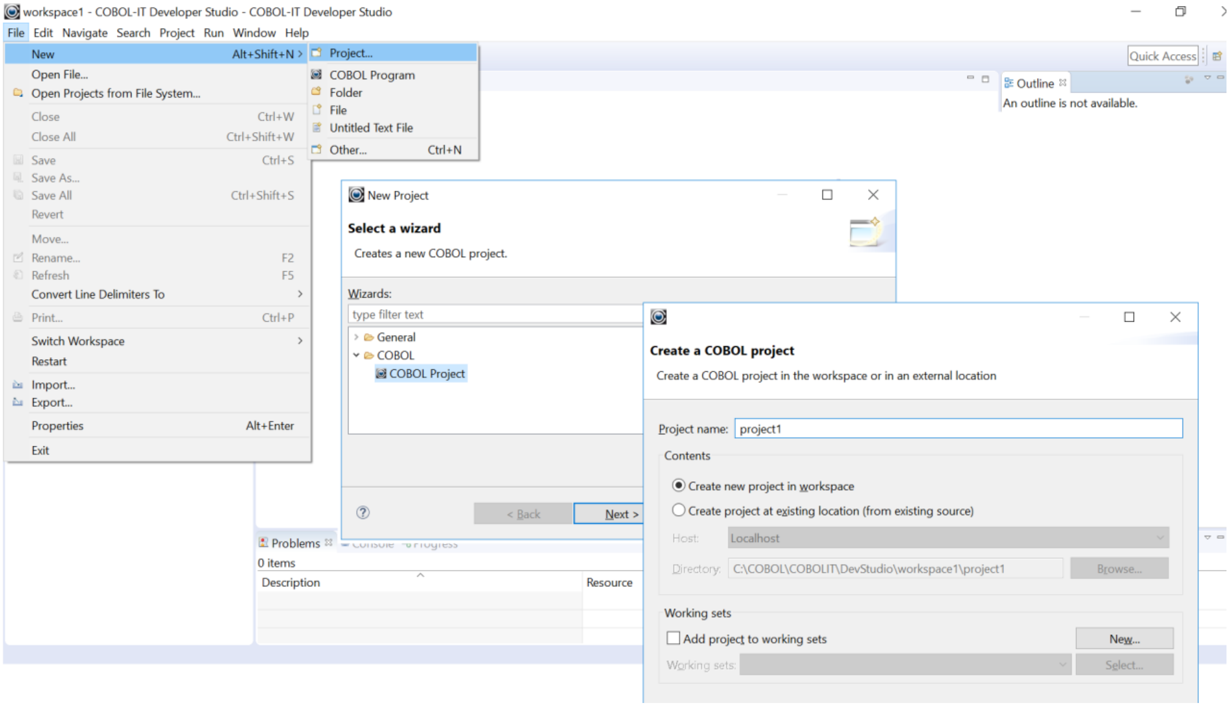
These settings control perspectives for running and debugging. A perspective can automatically open when launching or when an application suspends.

Open the associated perspective when launching
● Always ○ Never ○ Prompt

Open the associated perspective when an application suspends
○ Always ○ Never ● Prompt

These settings assign perspectives to each application type or launcher and launch mode set. Select "None" to indicate that a perspective should not be opened.

## Create a New COBOL Project



An outline is not available.

Creates a new COBOL project.

Create a COBOL project in the workspace or in an external location

Project name: project1

● Create new project in workspace
○ Create project at existing location (from existing source)

Host: Localhost

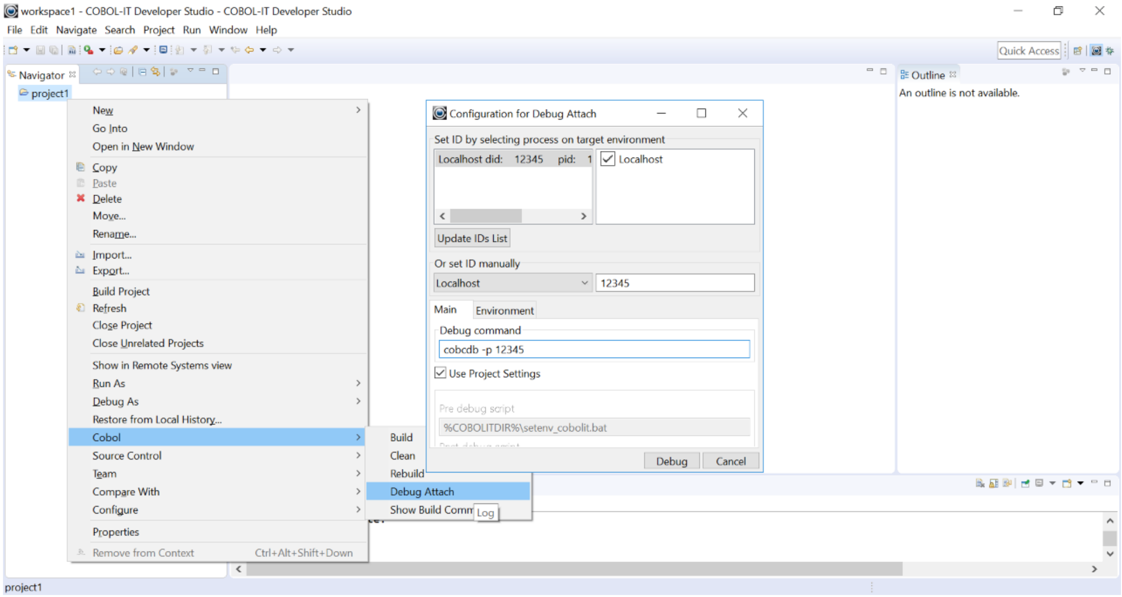
Directory: C:\COBOL\COBOLIT\DevStudio\workspace1\project1

□ Add project to working sets

Working sets:

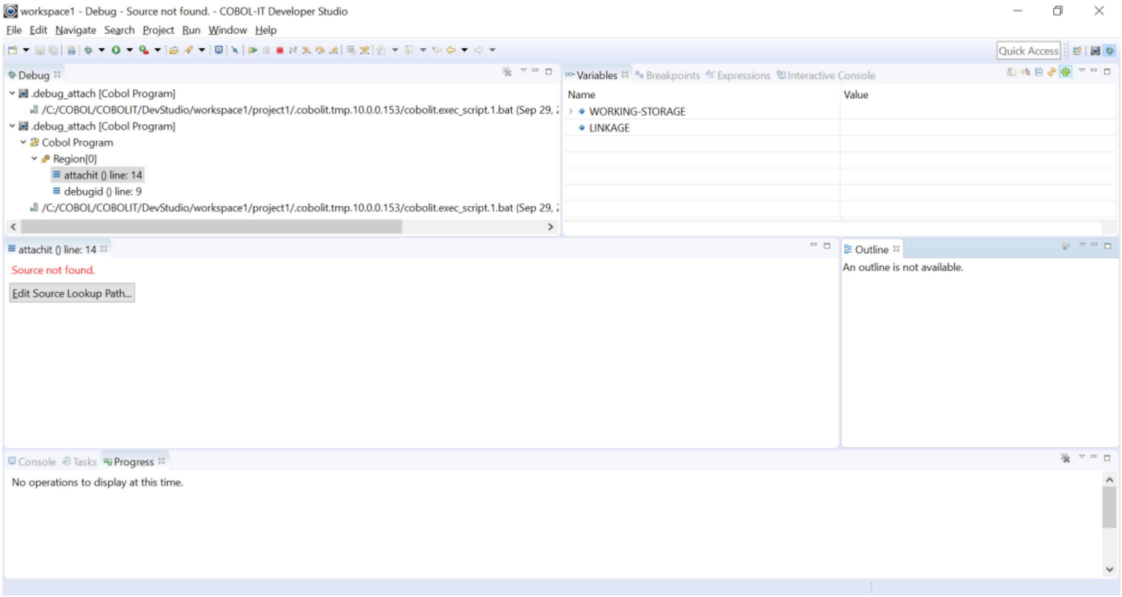
## Select Debug Attach Function

In the Navigator Window, right-click on the Project, and select COBOL>Debug Attach from the dropdown list. In the Debug Configuration for Reverse Attach Window, select the entry with the PID that matches the PID of the paused runtime session. Click “Debug”.

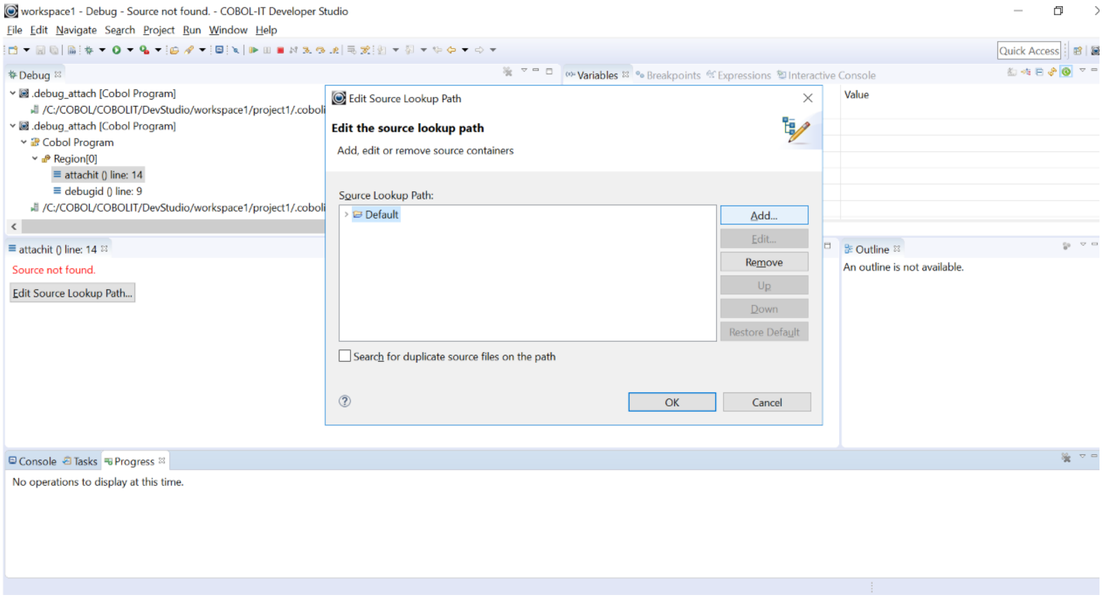


## Edit Source Lookup Path

The Developer Studio opens in the Debugger Perspective. Note that there is a message, in red, that Source Not Found. To associate the the source code of subpgm.cbl with the project, click on the Edit Source Lookup Path… button.


workspace1 - Debug - Source not found. - COBOL-IT Developer Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29,
.debug_attach [Cobol Program]
Cobol Program
Region[0]
attachit () line: 14
debugid () line: 9
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29,
Variables
Breakpoints
Expressions
Interactive Console
Name
Value
WORKING-STORAGE
LINKAGE
attachit () line: 14
Source not found.
Edit Source Lookup Path...
Outline
An outline is not available.
Console
Tasks
Progress
No operations to display at this time.

On the Edit the Source Lookup Path Screen, the Default setting is your current Project Path. If the source files are not in your Project Path ( they probably are not ), click the Add button.


workspace1 - Debug - Source not found. - COBOL-IT Developer Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.coboli
.debug_attach [Cobol Program]
Cobol Program
Region[0]
attachit () line: 14
debugid () line: 9
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.coboli
Variables
Breakpoints
Expressions
Interactive Console
Value
Edit Source Lookup Path
Add...
Edit...
Remove
Up
Down
Restore Default
OK
Cancel
Outline
attachit () line: 14
Edit Source Lookup Path...
Console
Tasks
Progress

Source not found.

Edit the source lookup path

Add, edit or remove source containers

Source Lookup Path:

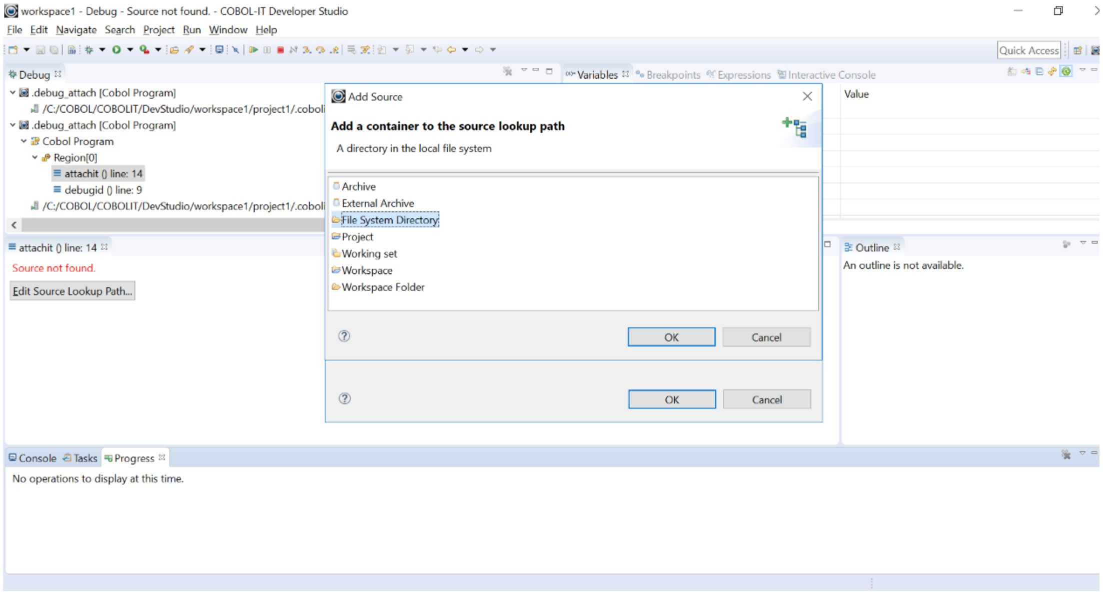
Default

□ Search for duplicate source files on the path

An outline is not available.

No operations to display at this time.

After clicking the Add button, select File System Directory, Click Ok.


workspace1 - Debug - Source not found. - COBOL-IT Developer Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.coboli
.debug_attach [Cobol Program]
Cobol Program
Region[0]
attachit () line: 14
debugid () line: 9
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.coboli
Variables
Breakpoints
Expressions
Interactive Console
Value
Add Source
OK
Cancel
OK
Cancel
Outline
attachit () line: 14
Edit Source Lookup Path...
Console
Tasks
Progress

Add a container to the source lookup path

A directory in the local file system

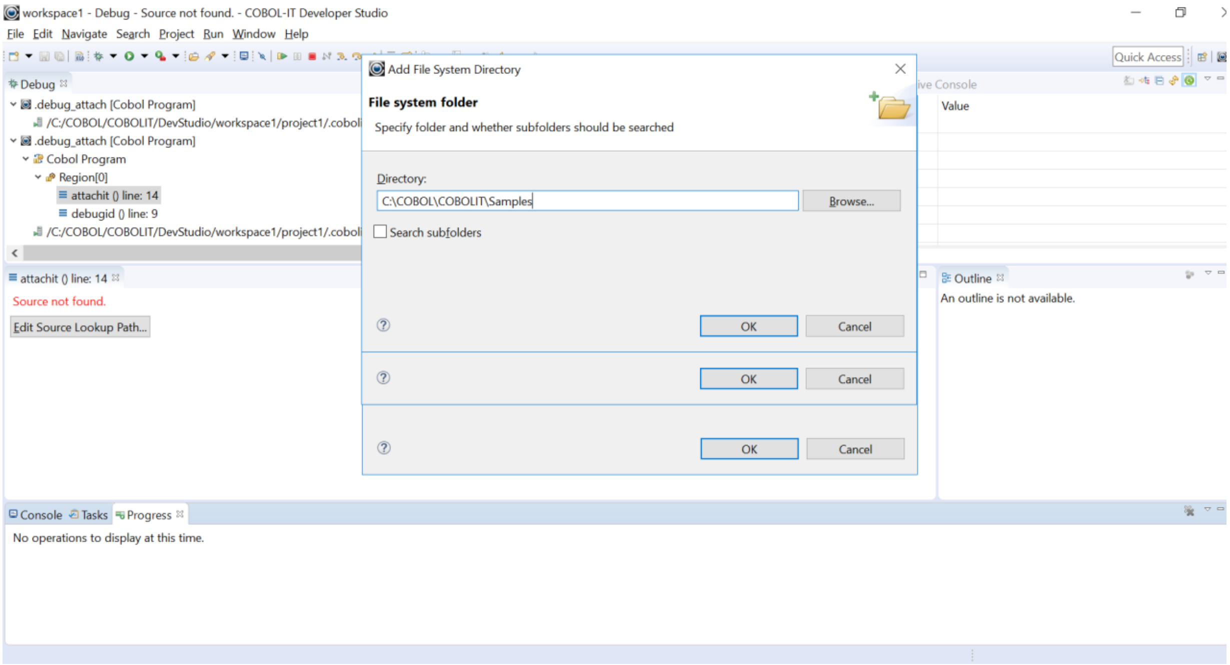
- Archive
- External Archive
- File System Directory
- Project
- Working set
- Workspace
- Workspace Folder

Source not found.

An outline is not available.

No operations to display at this time.

Use the Browse button to locate the Source Location


workspace1 - Debug - Source not found. - COBOL-IT Developer Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobol
.debug_attach [Cobol Program]
Cobol Program
Region[0]
attachit () line: 14
debugid () line: 9
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobol
attachit () line: 14
Edit Source Lookup Path...
Value
Outline
Console Tasks Progress
Browse...
OK
Cancel

Add File System Directory

**File system folder**

Specify folder and whether subfolders should be searched

Directory: C:\COBOL\COBOLIT\Samples

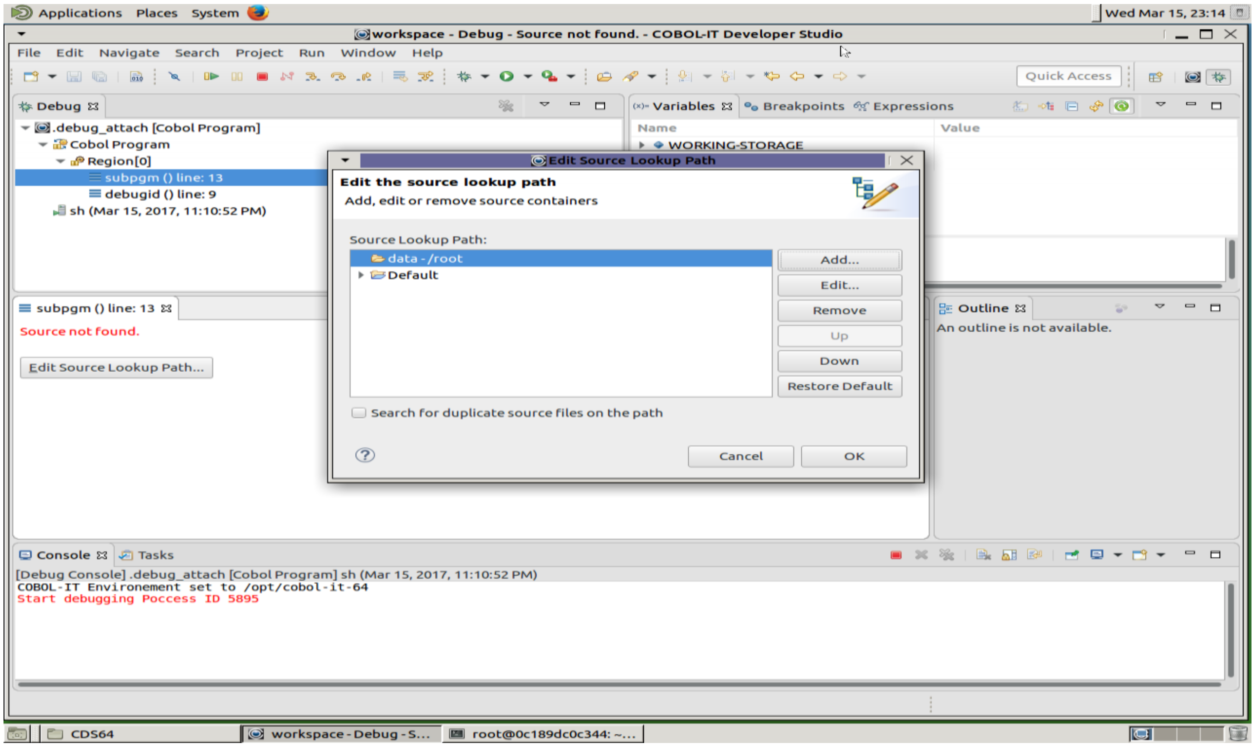
□ Search subfolders

Source not found.

An outline is not available.

No operations to display at this time.

Your selection will appear in the Edit Source Lookup Path window . Click OK.


Applications Places System
Wed Mar 15, 23:14
workspace - Debug - Source not found. - COBOL-IT Developer Studio
File Edit Navigate Search Project Run Window Help
Quick Access
Debug
Variables Breakpoints Expressions
Name
Value
WORKING-STORAGE
.debug_attach [Cobol Program]
Cobol Program
Region[0]
subpgm () line: 13
debugid () line: 9
sh (Mar 15, 2017, 11:10:52 PM)
subpgm () line: 13
Edit Source Lookup Path...
Outline
Add...
Edit...
Remove
Up
Down
Restore Default
Cancel
OK
Console Tasks
CDS64
workspace - Debug - S...
root@0c189dc0c344: ~...

Edit Source Lookup Path

**Edit the source lookup path**

Add, edit or remove source containers

Source Lookup Path:

- data - /root
- Default

□ Search for duplicate source files on the path

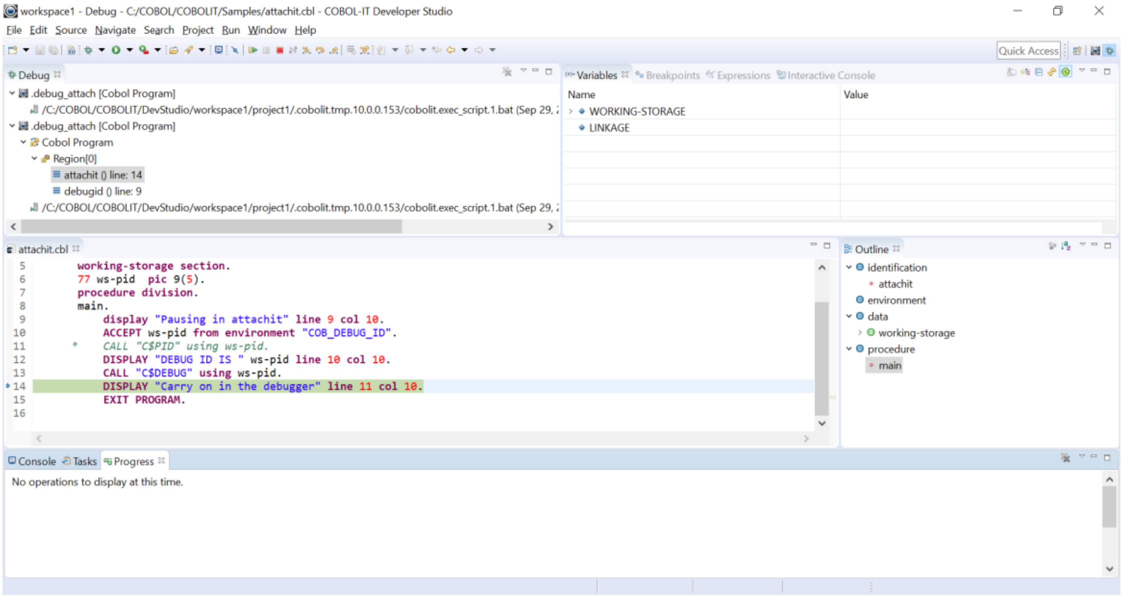
Source not found.

An outline is not available.

```
[Debug Console] .debug_attach [Cobol Program] sh (Mar 15, 2017, 11:10:52 PM)
COBOL-IT Environement set to /opt/cobol-it-64
Start debugging Poccess ID 5895
```

# Debug in the Developer Studio

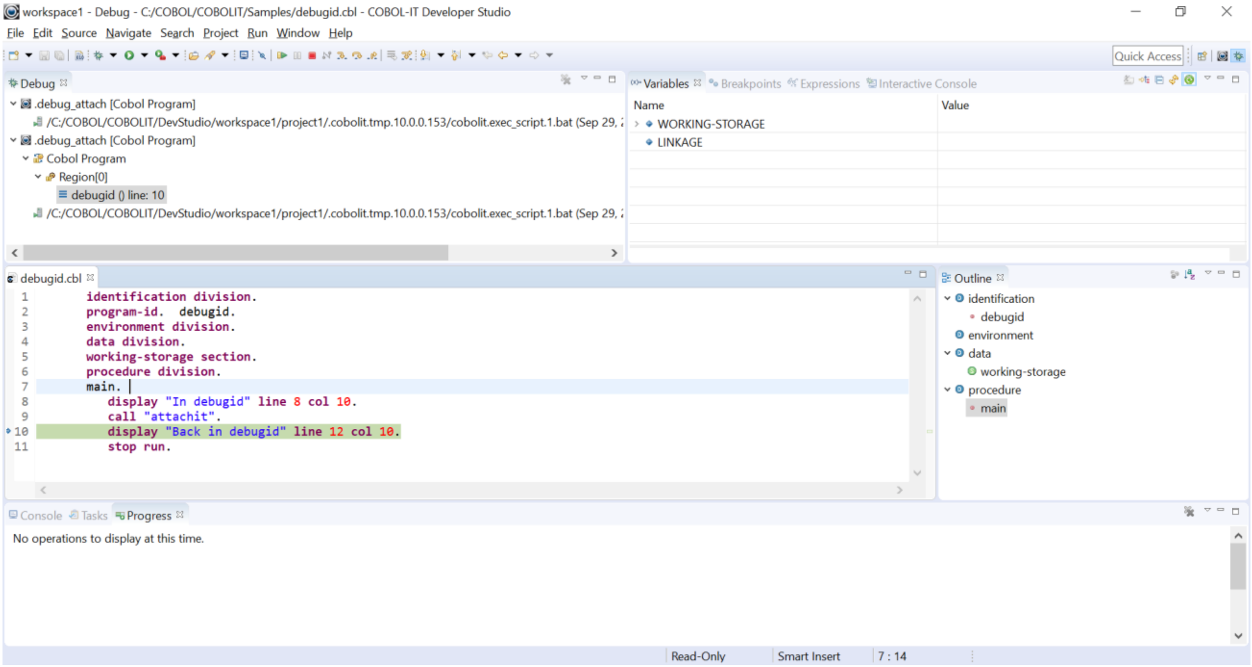
You are now ready to debug in the Developer Studio :


workspace1 - Debug - C:/COBOL/COBOLIT/Samples/attachit.cbl - COBOL-IT Developer Studio
File Edit Source Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2
.debug_attach [Cobol Program]
Cobol Program
Region[0]
attachit () line: 14
debugid () line: 9
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2
Variables
Breakpoints
Expressions
Interactive Console
Name
Value
WORKING-STORAGE
LINKAGE
attachit.cbl
Outline
identification
attachit
environment
data
working-storage
procedure
main
Console
Tasks
Progress

```
       working-storage section.
       77 ws-pid  pic 9(5).
       procedure division.
       main.
           display "Pausing in attachit" line 9 col 10.
           ACCEPT ws-pid from environment "COB_DEBUG_ID".
      *    CALL "C$PID" using ws-pid.
           DISPLAY "DEBUG ID IS " ws-pid line 10 col 10.
           CALL "C$DEBUG" using ws-pid.
           DISPLAY "Carry on in the debugger" line 11 col 10.
           EXIT PROGRAM.

```

No operations to display at this time.

Use the Debugger toolbar buttons to debug your program.


workspace1 - Debug - C:/COBOL/COBOLIT/Samples/debugid.cbl - COBOL-IT Developer Studio
File Edit Source Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2
.debug_attach [Cobol Program]
Cobol Program
Region[0]
debugid () line: 10
/C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29, 2
Variables
Breakpoints
Expressions
Interactive Console
Name
Value
WORKING-STORAGE
LINKAGE
debugid.cbl
Outline
identification
debugid
environment
data
working-storage
procedure
main
Console
Tasks
Progress
Read-Only
Smart Insert
7 : 14

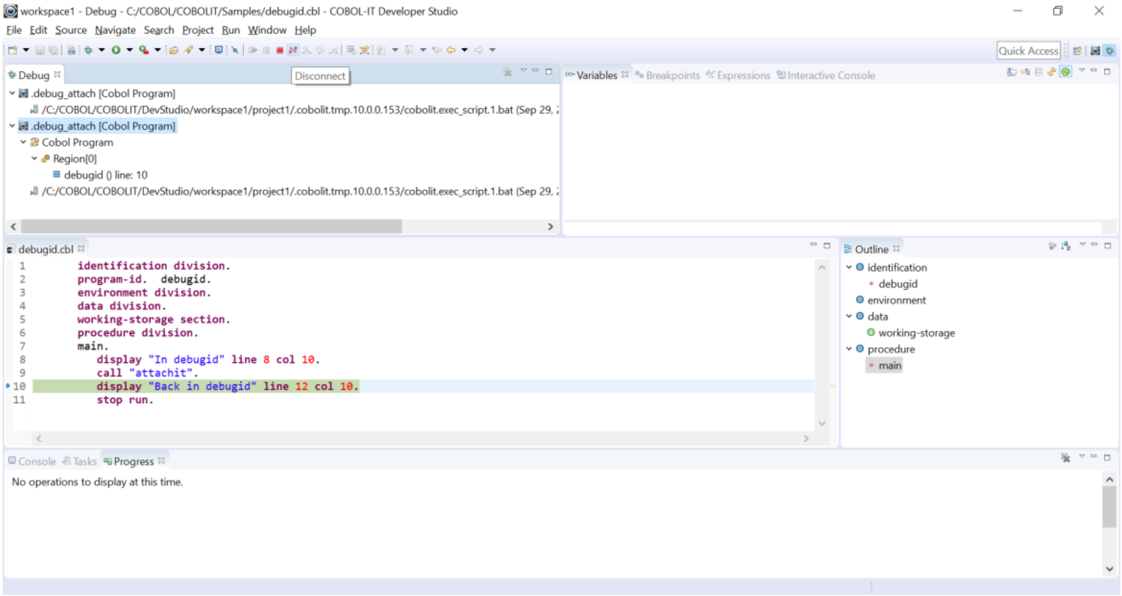
```
       identification division.
       program-id.  debugid.
       environment division.
       data division.
       working-storage section.
       procedure division.
       main.
          display "In debugid" line 8 col 10.
          call "attachit".
          display "Back in debugid" line 12 col 10.
          stop run.
```

No operations to display at this time.

## Disconnecting the Debugger

You may wish to disconnect the debugger and continue running your program.

To disconnect the debugger, select the highest level of your session in the debugging view, titled .debug-attach. This will enable the “Disconnect” toolbar button to the right of the Terminate button. Click on the Disconnect button to disconnect the COBOL debugger without terminating the runtime session.


workspace1 - Debug - C:/COBOL/COBOLIT/Samples/debugid.cbl - COBOL-IT Developer Studio
File Edit Source Navigate Search Project Run Window Help
Quick Access
Debug
Disconnect
Variables
Breakpoints
Expressions
Interactive Console
debugid.cbl
Outline
Console
Tasks
Progress

```
.debug_attach [Cobol Program]
    /C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29,
.debug_attach [Cobol Program]
  Cobol Program
    Region[0]
      debugid () line: 10
    /C:/COBOL/COBOLIT/DevStudio/workspace1/project1/.cobolit.tmp.10.0.0.153/cobolit.exec_script.1.bat (Sep 29,
```

```
       identification division.
       program-id.  debugid.
       environment division.
       data division.
       working-storage section.
       procedure division.
       main.
          display "In debugid" line 8 col 10.
          call "attachit".
          display "Back in debugid" line 12 col 10.
          stop run.
```

```
identification
  debugid
environment
data
  working-storage
procedure
  main
```

No operations to display at this time.

## Programs used in this sample

**debugid.cbl**

```
       identification division.
       program-id.  debugid.
       environment division.
       data division.
       working-storage section.
       procedure division.
       main.
          display "In debugid" line 8 col 10.
          call "attachit".
          display "Back in debugid" line 12 col 10.
          stop run.
```

**attachit.cbl**

```
        identification division.
```

```
       program-id.  attachit.
       environment division.
       data division.
       working-storage section.
       77 ws-pid  pic 9(5).
       procedure division.
       main.
           display "Pausing in attachit" line 9 col 10.
           ACCEPT ws-pid from environment "COB_DEBUG_ID".
      *    CALL "C$PID" using ws-pid.
           DISPLAY "DEBUG ID IS " ws-pid line 10 col 10.
           CALL "C$DEBUG" using ws-pid.
           DISPLAY "Carry on in the debugger" line 11 col 10.
           EXIT PROGRAM.
```

**runit.bat**

```
SET COB_DEBUG_ID=12345
SET COB_LIBRARY_PATH=C:\COBOL\COBOLIT\Samples
cobc -g debugid.cbl
cobc -g attachit.cbl
cobcrun debugid
```

# Attaching a “C” debugger to a running process

## -G

Produces debugging information in the output, allowing “C”-level debugging.

To perform “C” level debugging, use the COBOL-IT Developer Studio.

COBOL-IT translates COBOL to “C” and uses the host “C” compiler to compile the translated source.  As preparation, compile your COBOL programs with the –G compiler flag.  “C” modules should be compiled for debugging as well.

The –G COBOL compiler flag allows the COBOL program to be include information for the “C” debugger.  This corresponds to the gcc –g compiler flag.

Using the Debug Attach functionality of the Developer Studio to Attach the COBOL Debugger to an  Application, you can enter the COBOL Debugger, then start the “C” debugger, and proceed your debuggeing with both the “C” and COBOL debuggers running.

The Eclipse IDE for C/C++ Developers, and “C” compiler are required for this exercise.

## -fnostrip

Causes objects and object and executable files to NOT be stripped.
Stripping an object or an executable is the action of removing system level debugging information. The –fno-strip compiler flag is enabled by the –G compiler flag.

## Overview

Attaching a "C" Debugger to an Application is a functionality provided by Eclipse, when the Eclipse IDE for C/C++ Developers plug-in is installed, and the gdb debugger is installed on your system.

As preparation, compile your COBOL programs with the –G compiler flag. The –G COBOL compiler flag allows the COBOL program to be include information for the "C" debugger. This corresponds to the gcc –g compiler flag. "C" modules should be compiled for debugging as well.

We will create an executable called "newcall", linking a "C" library compiled for debugging with a COBOL program compiled to allow for "C" debugging.

We will start our executable from the command-line, use the Debug Attach functionality of the Developer Studio to Attach the COBOL Debugger to the Application.

At that point, we will start the "C" debugger, and continue with both the COBOL and "C" debuggers running. We will enter a CALL'ed "C" function, and use the "C" debugger. When finished debugging, we can return to the COBOL program, and resume using the COBOL debugger.

## Pre-requisites

The following must be installed on your Linux machine:

COBOL-IT Compiler Suite Enterprise Edition
COBOL-IT Developer Studio
C/C++ Development Tooling
The gdb source level debugger

## Sample Programs

**`c_printf.c`**

c_printf is called from our main COBOL program.
We will switch to the "C" debugger before executing the CALL of this function.

```
int c_printf (char * format , char * var)
{
     printf (format, var);
     return 0;
}
```

**newcall.cbl**

The  CALL “C$DEBUG” statement allows us to attach the COBOL debugger to a process.  After attaching the COBOL debugger to the process, we will then switch to the “C” debugger for the CALL to c_printf.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. NEWCALL.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
     01 ws-dummy PIC x.
     PROCEDURE DIVISION .
     MAIN.
         DISPLAY "NEWCALL Started"
         CALL "C$DEBUG" USING 12345
         DISPLAY "The PROGRAM will pause here" line  10 col 10.
         CALL "c_printf" USING "%s" "Hello"
         DISPLAY "Set another breakpoint here" line 12 col 10.
         EXIT PROGRAM.
```

## Compile and link the sample programs

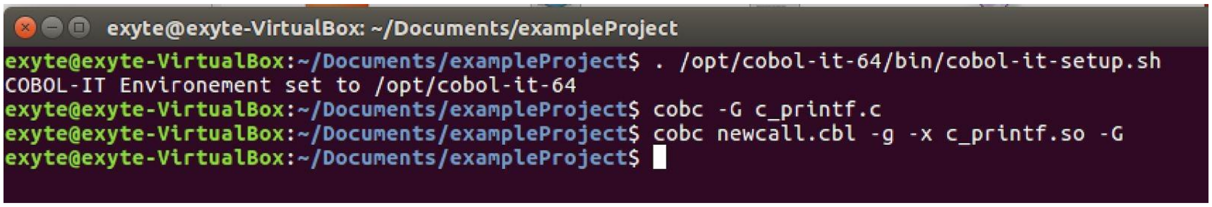
Compile c_printf.c with debugging information, and create a shared library called “clibs.so”. As a shortcut, you can use cobc to compile the “C” program. In this case, use the –G compiler flag.

```
>cobc –G c_printf.c
```

c_printf.so is compiled with debugging information.

Compile newcall.cbl with COBOL debugging information (-g) and C debugging information (-G).  Link c_printf.so and create an executable (-x) called newcall.

```
> cobc newcall.cbl –g –x c_printf.so –G
```



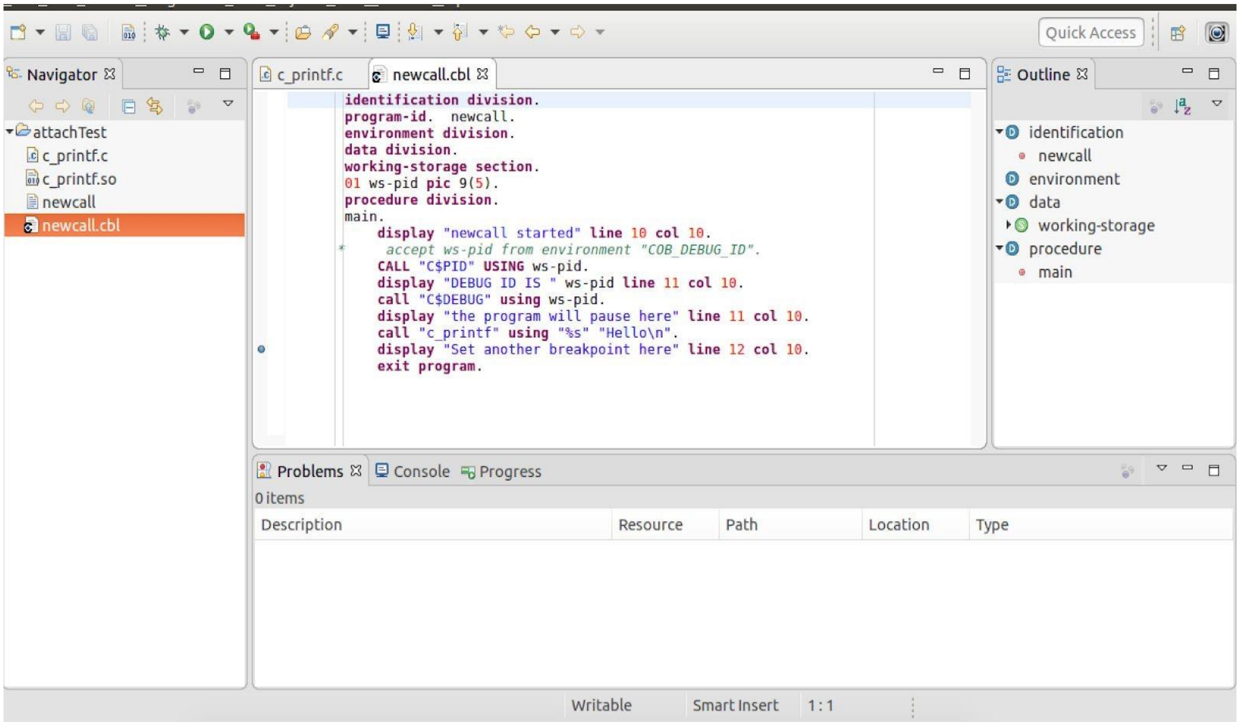
```
exyte@exyte-VirtualBox: ~/Documents/exampleProject
exyte@exyte-VirtualBox:~/Documents/exampleProject$ . /opt/cobol-it-64/bin/cobol-it-setup.sh
COBOL-IT Environement set to /opt/cobol-it-64
exyte@exyte-VirtualBox:~/Documents/exampleProject$ cobc -G c_printf.c
exyte@exyte-VirtualBox:~/Documents/exampleProject$ cobc newcall.cbl -g -x c_printf.so -G
exyte@exyte-VirtualBox:~/Documents/exampleProject$
```

This creates an executable module called newcall, in which c_printf.so is linked to the main module.

## Create a New COBOL project

Drag and drop into the project folder: c_printf.c, c_printf.so, newcall.cbl and the executable newcall.

Open newcall.cbl in the code editor and set a breakpoint:



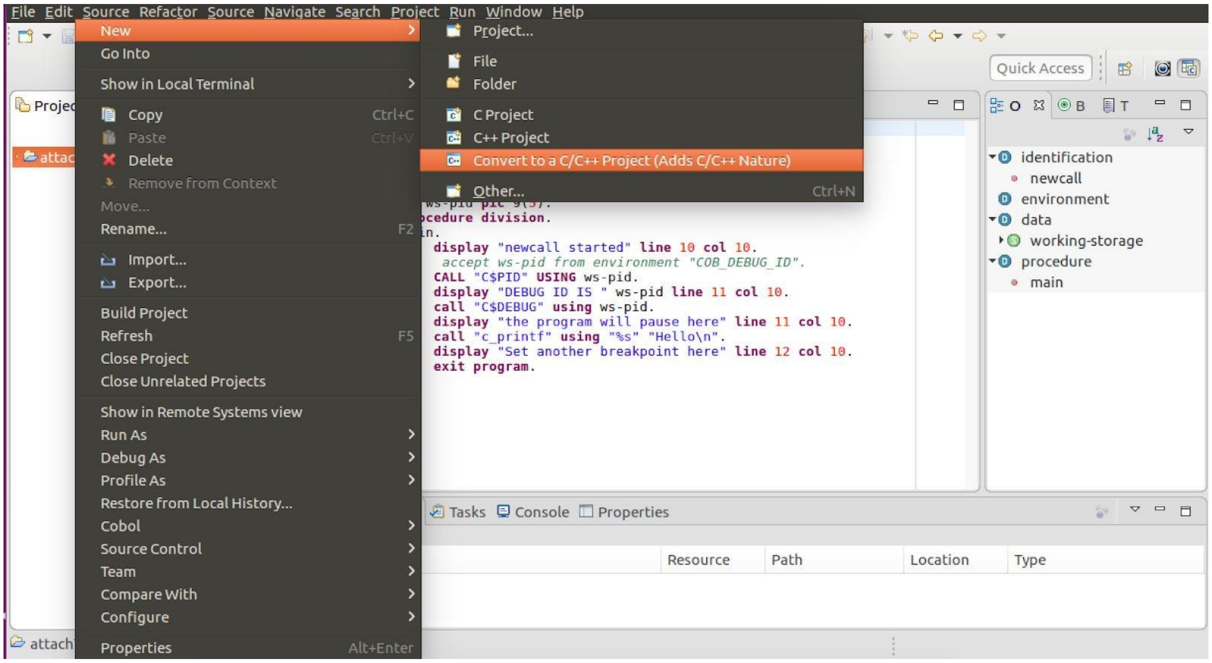
```
identification division.
program-id.  newcall.
environment division.
data division.
working-storage section.
01 ws-pid pic 9(5).
procedure division.
main.
    display "newcall started" line 10 col 10.
*    accept ws-pid from environment "COB_DEBUG_ID".
    CALL "C$PID" USING ws-pid.
    display "DEBUG ID IS " ws-pid line 11 col 10.
    call "C$DEBUG" using ws-pid.
    display "the program will pause here" line 11 col 10.
    call "c_printf" using "%s" "Hello\n".
    display "Set another breakpoint here" line 12 col 10.
    exit program.
```

Image 1- The COBOL Project, with files and breakpoint

## Add C/C++ nature to the project

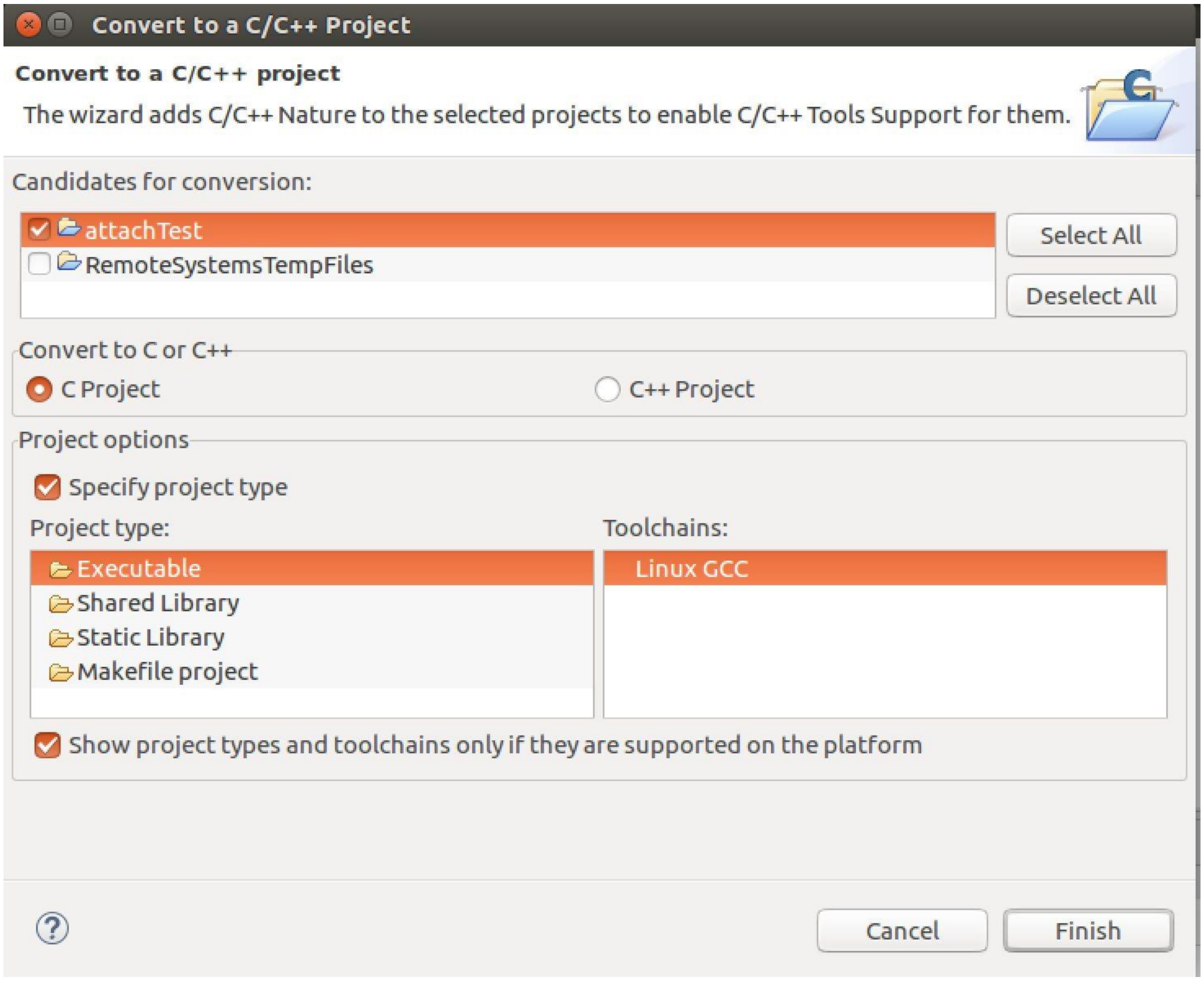
Add C/C++ nature to the project. The project needs to be both a COBOL Cobol and C/C++ project at the same time.

In the C/C++ Perspective , select File>New>Convert to a C/C++ Project.



```
 ws-pid pic 9(5).
ocedure division.
in.
  display "newcall started" line 10 col 10.
   accept ws-pid from environment "COB_DEBUG_ID".
  CALL "C$PID" USING ws-pid.
  display "DEBUG ID IS " ws-pid line 11 col 10.
  call "C$DEBUG" using ws-pid.
  display "the program will pause here" line 11 col 10.
  call "c_printf" using "%s" "Hello\n".
  display "Set another breakpoint here" line 12 col 10.
  exit program.
```

This will open the Convert to C/C++ Project Wizard:



Convert to a C/C++ Project

**Convert to a C/C++ project**

The wizard adds C/C++ Nature to the selected projects to enable C/C++ Tools Support for them.

Candidates for conversion:

- ☑ attachTest
- ☐ RemoteSystemsTempFiles

Convert to C or C++

● C Project ○ C++ Project

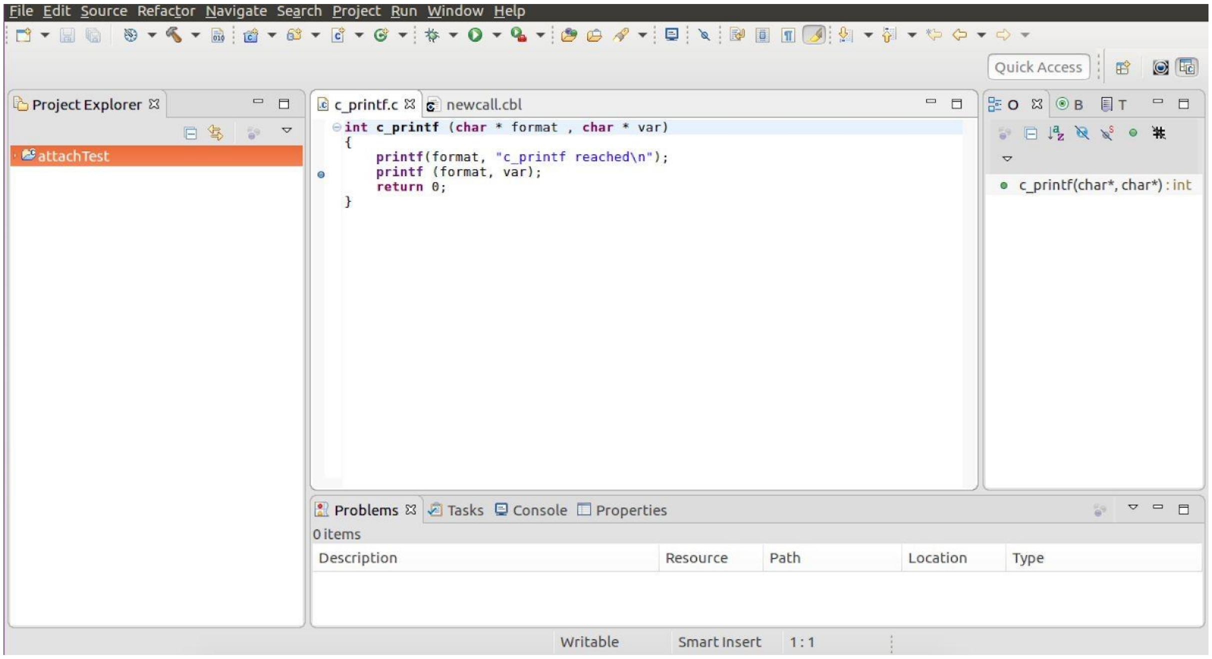
Project options

☑ Specify project type

| Project type: | Toolchains: |
|---|---|
| Executable | Linux GCC |
| Shared Library | |
| Static Library | |
| Makefile project | |

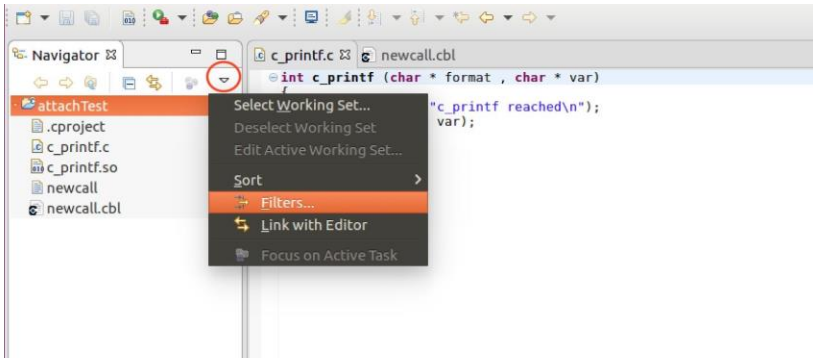
☑ Show project types and toolchains only if they are supported on the platform

In the C/C++ Perspective, open the C file and set a breakpoint:

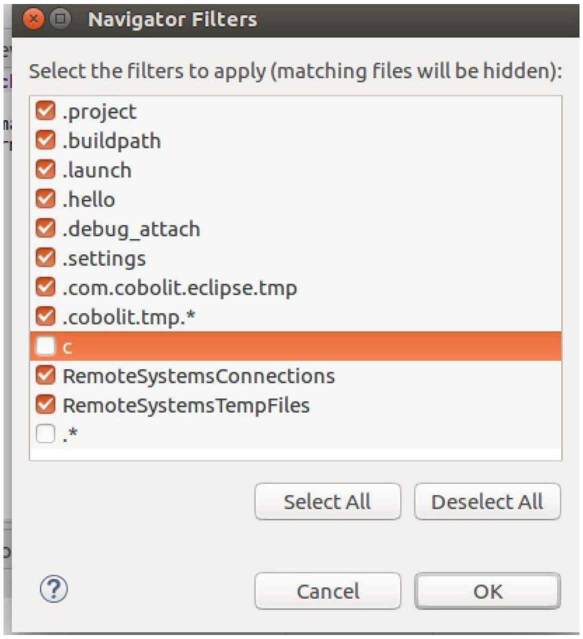


```
int c_printf (char * format , char * var)
{
    printf(format, "c_printf reached\n");
    printf (format, var);
    return 0;
}
```

In the COBOL Perspective, in the navigator window, configure the visibility of the “c” folder by de-selecting it:



```
int c_printf (char * format , char * var)
"c_printf reached\n");
var);
```



Select the filters to apply (matching files will be hidden):

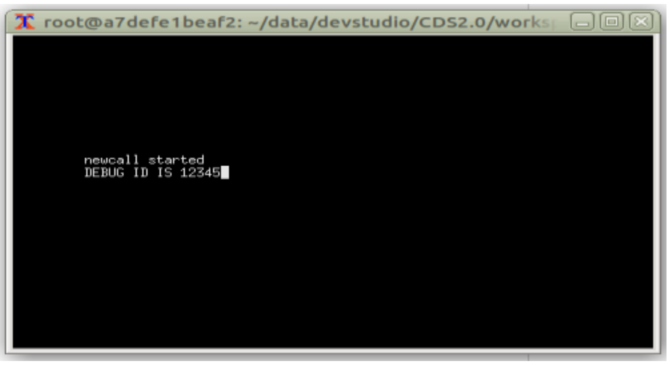
- ☑ .project
- ☑ .buildpath
- ☑ .launch
- ☑ .hello
- ☑ .debug_attach
- ☑ .settings
- ☑ .com.cobolit.eclipse.tmp
- ☑ .cobolit.tmp.*
- □ c
- ☑ RemoteSystemsConnections
- ☑ RemoteSystemsTempFiles
- □ .*

## Launch the newcall executable

Run newcall from the command-line

```
> ./newcall
```

newcall pauses after the CALL C$DEBUG command.



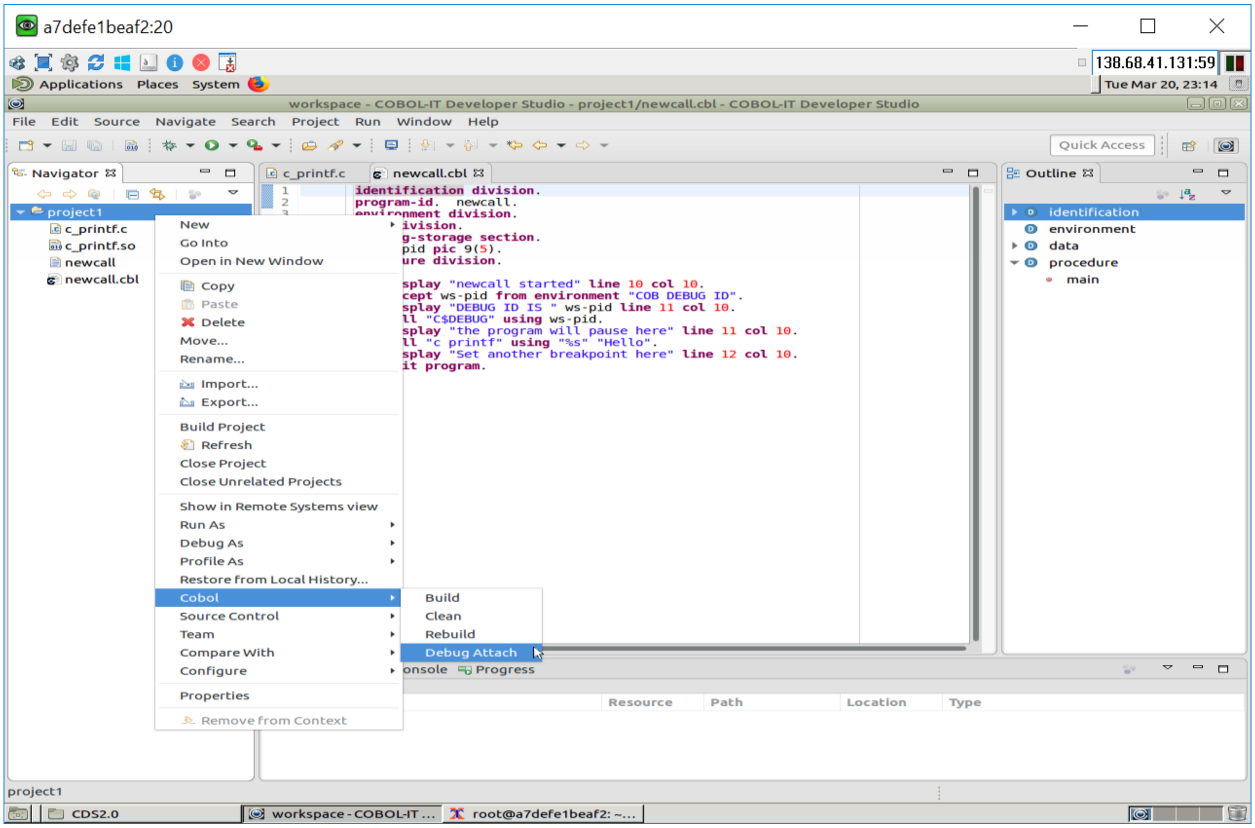
```
newcall started
DEBUG ID IS 12345
```

Image 2- Newcall is paused

At this point, re-enter the Developer Studio.

# The Debug Attach function

Right-click on the Project name, and select Cobol from the menu.
Then Select Debug Attach.

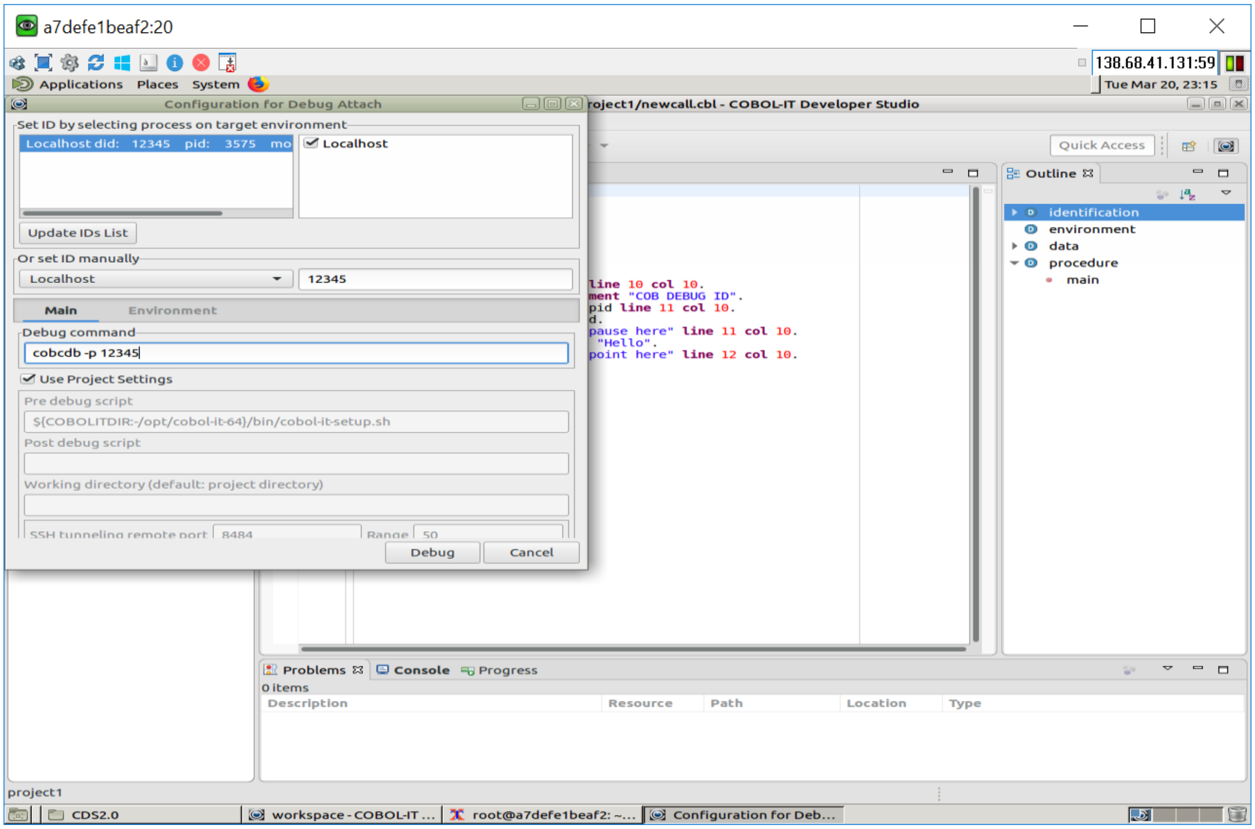


```
identification division.
program-id.  newcall.
environment division.
ivision.
g-storage section.
pid pic 9(5).
ure division.

splay "newcall started" line 10 col 10.
cept ws-pid from environment "COB DEBUG ID".
splay "DEBUG ID IS " ws-pid line 11 col 10.
ll "C$DEBUG" using ws-pid.
splay "the program will pause here" line 11 col 10.
ll "c printf" using "%s" "Hello".
splay "Set another breakpoint here" line 12 col 10.
it program.
```

Image 3

## The Reverse Attach Window

From the Reverse Attach Window, select Newcall.
Click on the Debug button.



a7defe1beaf2:20

Configuration for Debug Attach

Set ID by selecting process on target environment
Localhost did: 12345 pid: 3575 mo
☑ Localhost
Update IDs List

Or set ID manually
Localhost ▾ 12345

Main | Environment

Debug command
cobcdb -p 12345

☑ Use Project Settings
Pre debug script
${COBOLITDIR:-/opt/cobol-it-64}/bin/cobol-it-setup.sh
Post debug script

Working directory (default: project directory)

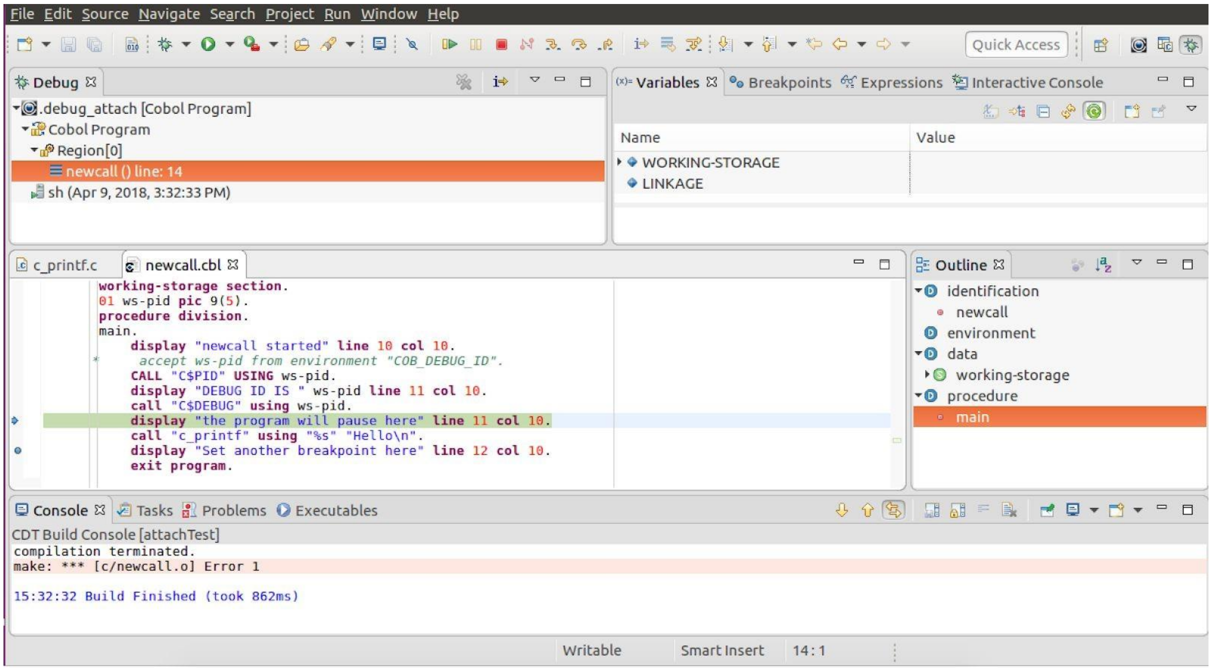
SSH tunneling remote port 8484 Range 50

Debug | Cancel

Image 4- The Reverse Attach Windows

## The COBOL-IT Debugger Attach

Since the source file is in the project, newcall is opened directly into the COBOL-IT Debugger Perspective.



```
       working-storage section.
       01 ws-pid pic 9(5).
       procedure division.
       main.
           display "newcall started" line 10 col 10.
      *     accept ws-pid from environment "COB_DEBUG_ID".
           CALL "C$PID" USING ws-pid.
           display "DEBUG ID IS " ws-pid line 11 col 10.
           call "C$DEBUG" using ws-pid.
           display "the program will pause here" line 11 col 10.
           call "c_printf" using "%s" "Hello\n".
           display "Set another breakpoint here" line 12 col 10.
           exit program.
```

CDT Build Console [attachTest]

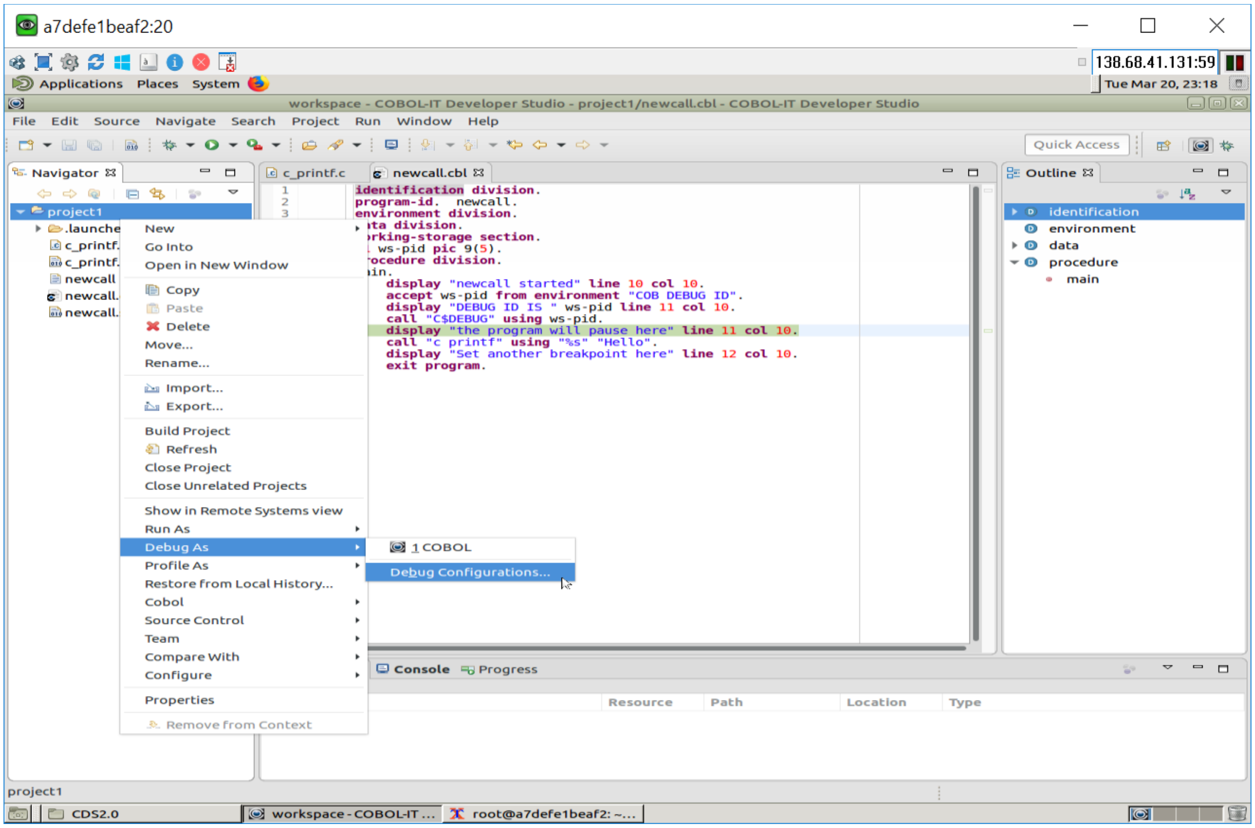
```
compilation terminated.
make: *** [c/newcall.o] Error 1

15:32:32 Build Finished (took 862ms)
```

Image 5- The COBOL-IT debugger is running.

## C/C++ Debug Configuration

To start the "C" debugger, return to the Developer Studio Perspective. In the Navigator Window, select the Project by clicking on it. Right-click on the Project to open the drop-down menu. From the drop-down menu, select Debug As, and then the Debug Configurations function.

The Debug Configurations contain the interfaces to configure the C/C++ Attach to Application Functionality.

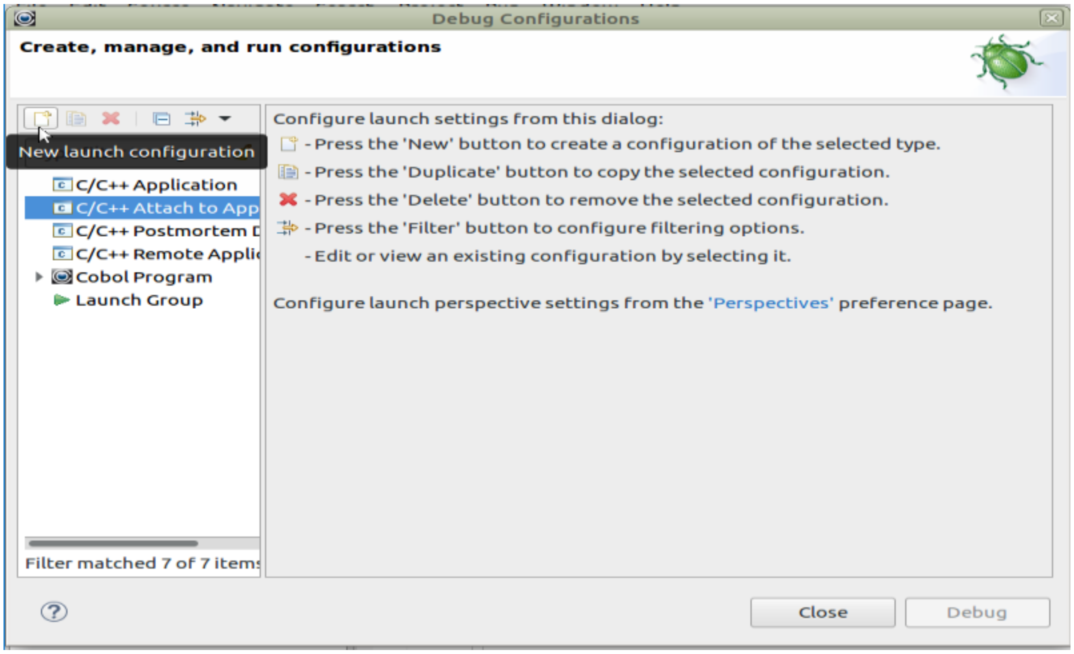


```
identification division.
program-id.  newcall.
environment division.
ta division.
rking-storage section.
 ws-pid pic 9(5).
ocedure division.
in.
    display "newcall started" line 10 col 10.
    accept ws-pid from environment "COB DEBUG ID".
    display "DEBUG ID IS " ws-pid line 11 col 10.
    call "C$DEBUG" using ws-pid.
    display "the program will pause here" line 11 col 10.
    call "c printf" using "%s" "Hello".
    display "Set another breakpoint here" line 12 col 10.
    exit program.
```

Image 6- Select Debug As/Debug Configurations

## Create a new C/C++ Attach to Application

Select C/C++ Attach to Application by clicking on it.  Click on the “New” button on the toolbar above the panel on the left to create a new configuration.



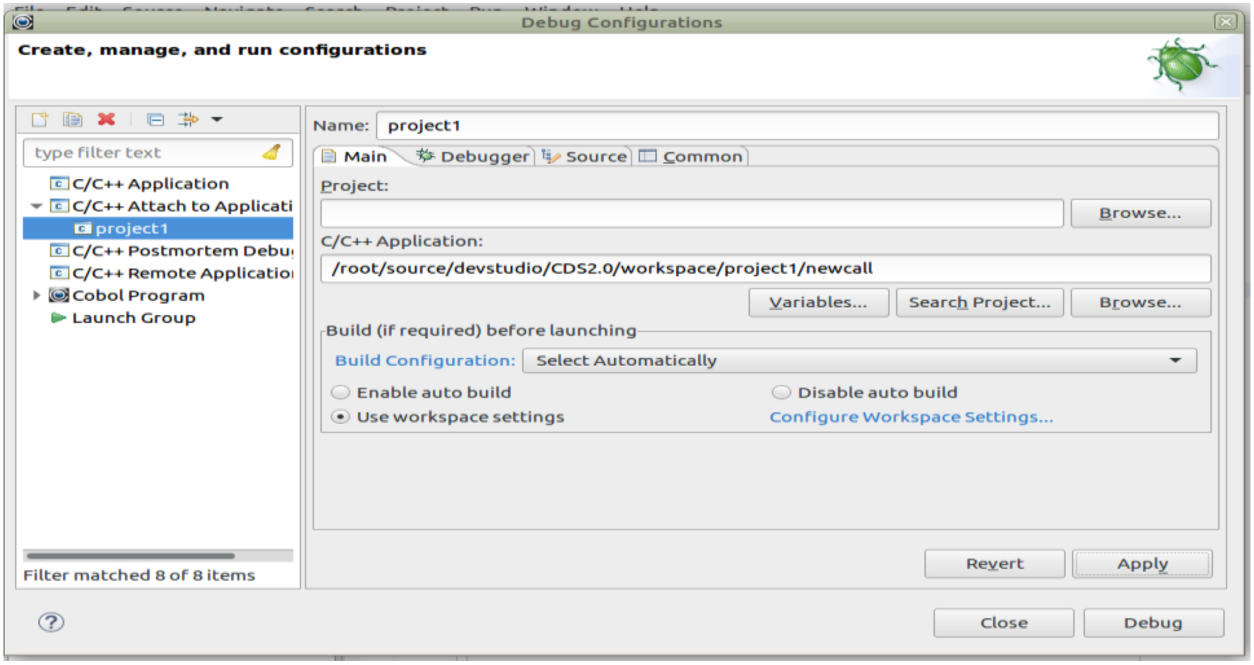
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Image 7- New C/C++ Attach to Application

## C/C++ Debug Configuration- Main Tab

The Name of your configuration is pre-filled.
Enter the full path, and name of your C/C++ Application.
In our case, this is the newcall application.



Debug Configurations

Create, manage, and run configurations

Name: project1

Project:

C/C++ Application: /root/source/devstudio/CDS2.0/workspace/project1/newcall

Build (if required) before launching

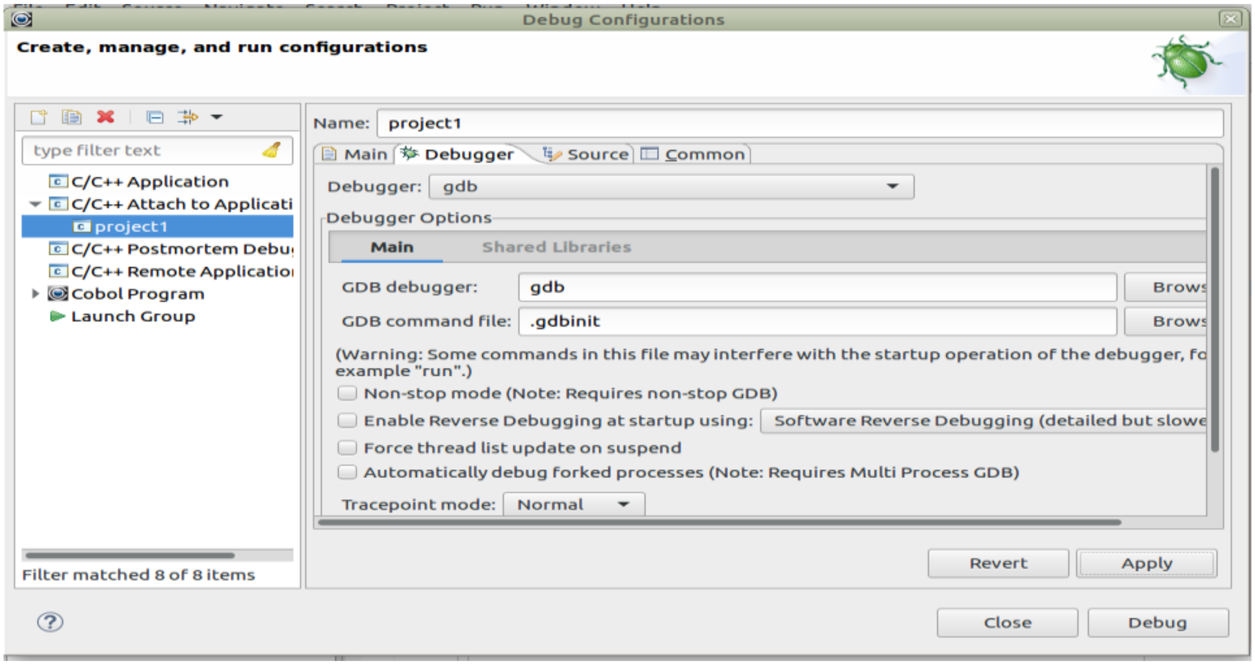
Build Configuration: Select Automatically

○ Enable auto build
○ Disable auto build
● Use workspace settings

Image 8- The Main Tab

## C/C++ Debug Configuration- Debugger Tab

No changes need to be made on the Debugger tab.
Note that gdb is named as the debugger.
gdb must be installed on your Linux machine.



Name: project1

Debugger: gdb

GDB debugger: gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, fo example "run".)

☐ Non-stop mode (Note: Requires non-stop GDB)

☐ Enable Reverse Debugging at startup using: Software Reverse Debugging (detailed but slowe

☐ Force thread list update on suspend

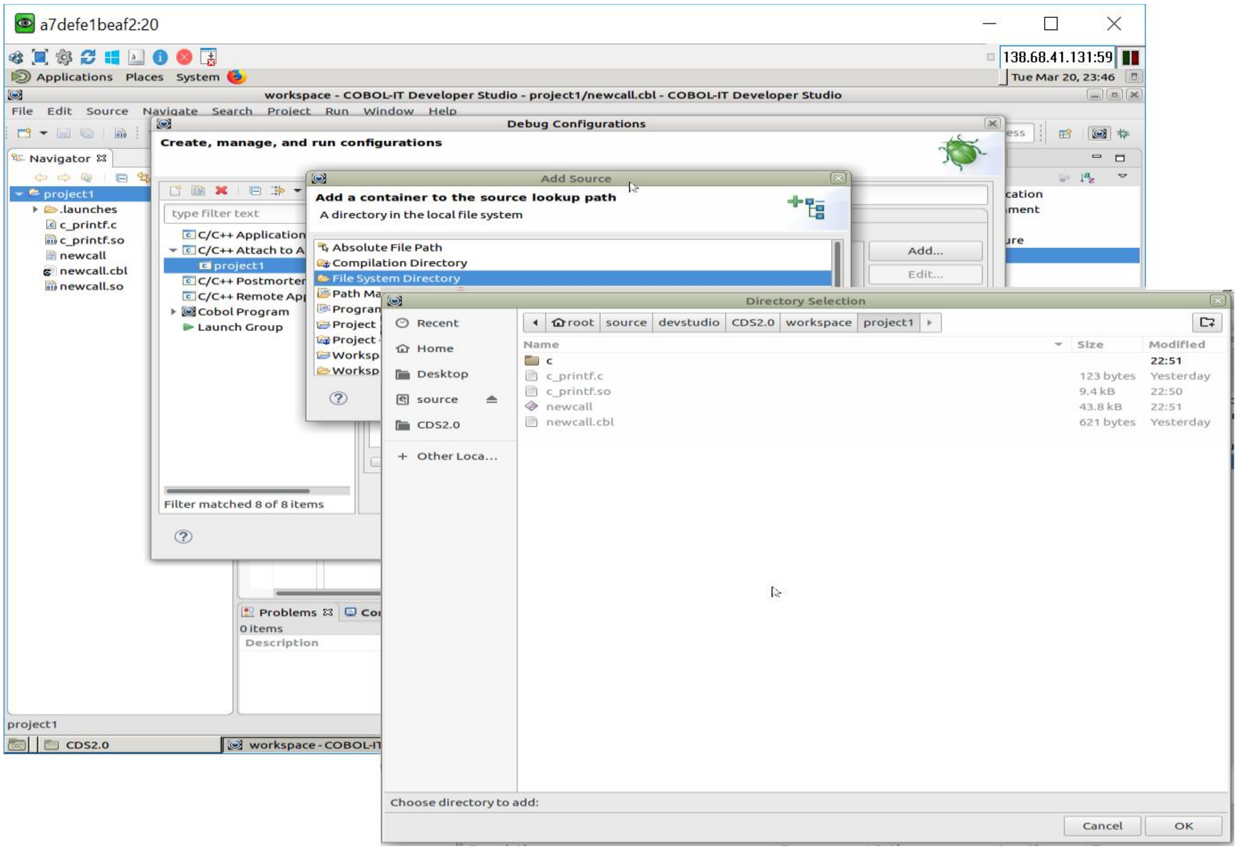
☐ Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Image 9- The Debugger Tab

## C/C++ Debug Configuration- Source Tab

Click on the Add button. In the Add Source Windows, select “File System Directory”. Click Ok. On the Directory Selection screen, select project1. Click Ok.

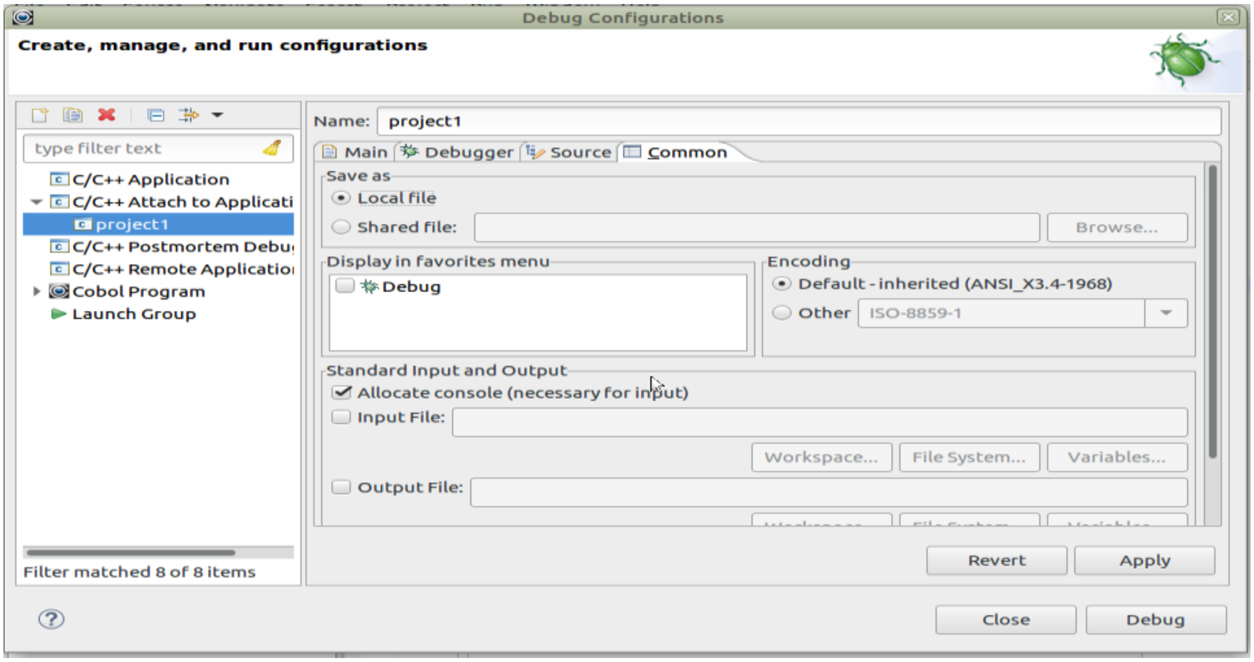


| Name | Size | Modified |
|---|---|---|
| c | | 22:51 |
| c_printf.c | 123 bytes | Yesterday |
| c_printf.so | 9.4 kB | 22:50 |
| newcall | 43.8 kB | 22:51 |
| newcall.cbl | 621 bytes | Yesterday |

Image 10- The Source Tab

## C/C++ Debug Configuration- Common Tab

No changes need to be made on the Common tab.
Select Apply. Click on the Debug button to enter the Debugger.



Name: project1

Save as
● Local file
○ Shared file:

Display in favorites menu
□ Debug

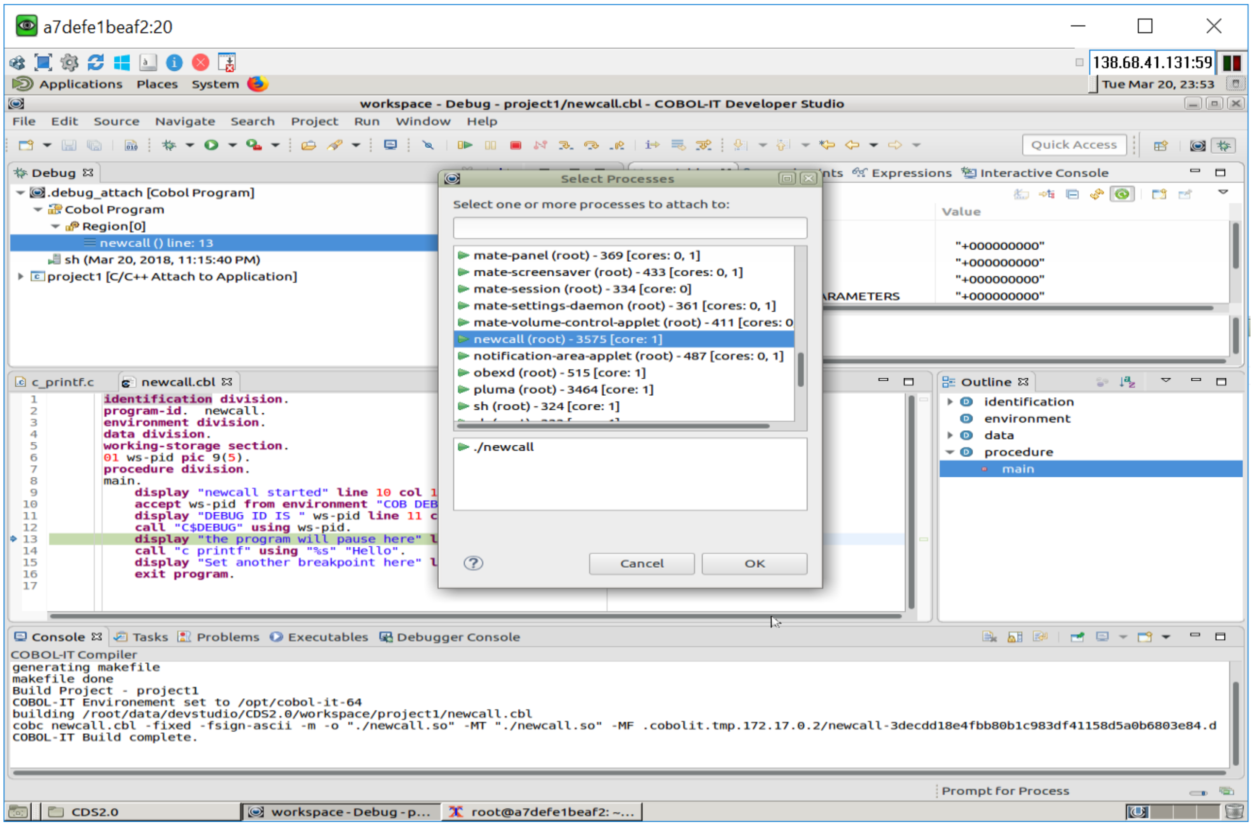
Encoding
● Default - inherited (ANSI_X3.4-1968)
○ Other ISO-8859-1

Standard Input and Output
☑ Allocate console (necessary for input)
□ Input File:
□ Output File:

Image 11- The Common Tab

## Select newcall from list of running applications

The system opens a list of running applications. Search by name or PID. Select newcall, click Ok. Control returns to the debug window.



Select Processes

Select one or more processes to attach to:

- mate-panel (root) - 369 [cores: 0, 1]
- mate-screensaver (root) - 433 [cores: 0, 1]
- mate-session (root) - 334 [core: 0]
- mate-settings-daemon (root) - 361 [cores: 0, 1]
- mate-volume-control-applet (root) - 411 [cores: 0
- newcall (root) - 3575 [core: 1]
- notification-area-applet (root) - 487 [cores: 0, 1]
- obexd (root) - 515 [core: 1]
- pluma (root) - 3464 [core: 1]
- sh (root) - 324 [core: 1]

- ./newcall

Cancel OK

```
identification division.
program-id.  newcall.
environment division.
data division.
working-storage section.
01 ws-pid pic 9(5).
procedure division.
main.
    display "newcall started" line 10 col 1
    accept ws-pid from environment "COB DEB
    display "DEBUG ID IS " ws-pid line 11 c
    call "C$DEBUG" using ws-pid.
    display "the program will pause here" l
    call "c printf" using "%s" "Hello".
    display "Set another breakpoint here" l
    exit program.
```

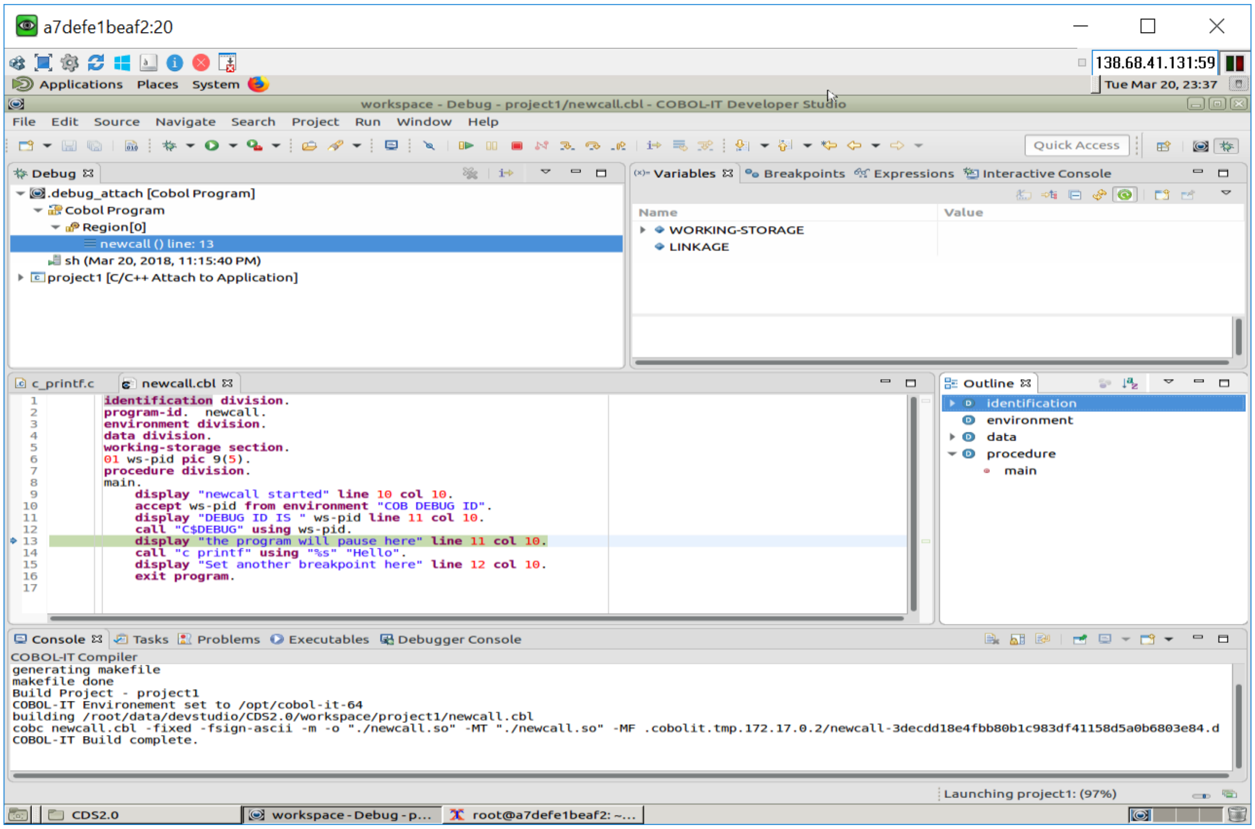
COBOL-IT Compiler

```
generating makefile
makefile done
Build Project - project1
COBOL-IT Environement set to /opt/cobol-it-64
building /root/data/devstudio/CDS2.0/workspace/project1/newcall.cbl
cobc newcall.cbl -fixed -fsign-ascii -m -o "./newcall.so" -MT "./newcall.so" -MF .cobolit.tmp.172.17.0.2/newcall-3decdd18e4fbb80b1c983df41158d5a0b6803e84.d
COBOL-IT Build complete.
```

Image 12- Search and select from the list of running applications

## The Debugger Window- Two Debugging Sessions

There are 2 debugging sessions in the Debug Window. One of the sessions is the COBOL debugger, and the other is the “C” debugger.  You are debugging the same application, but some of the lines are being tracked by the COBOL debugger, and some by the “C” debugger.


a7defe1beaf2:20
138.68.41.131:59
Applications Places System
Tue Mar 20, 23:37
workspace - Debug - project1/newcall.cbl - COBOL-IT Developer Studio
File Edit Source Navigate Search Project Run Window Help
Quick Access
Debug
.debug_attach [Cobol Program]
Cobol Program
Region[0]
newcall () line: 13
sh (Mar 20, 2018, 11:15:40 PM)
project1 [C/C++ Attach to Application]
Variables Breakpoints Expressions Interactive Console
Name
Value
WORKING-STORAGE
LINKAGE
c_printf.c
newcall.cbl
Outline
identification
environment
data
procedure
main
Console Tasks Problems Executables Debugger Console
COBOL-IT Compiler
Launching project1: (97%)
CDS2.0
workspace - Debug - p...
root@a7defe1beaf2: ~...

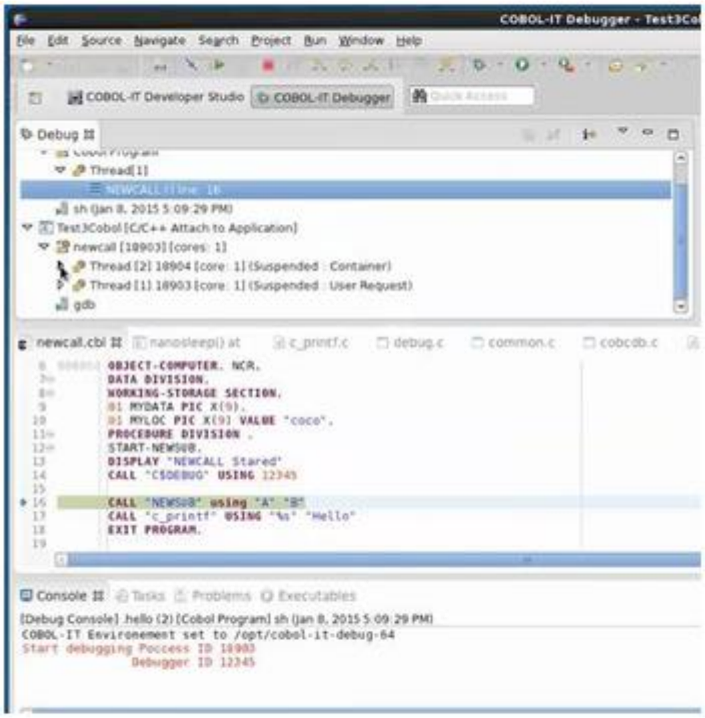
```
identification division.
program-id.  newcall.
environment division.
data division.
working-storage section.
01 ws-pid pic 9(5).
procedure division.
main.
    display "newcall started" line 10 col 10.
    accept ws-pid from environment "COB DEBUG ID".
    display "DEBUG ID IS " ws-pid line 11 col 10.
    call "C$DEBUG" using ws-pid.
    display "the program will pause here" line 11 col 10.
    call "c printf" using "%s" "Hello".
    display "Set another breakpoint here" line 12 col 10.
    exit program.
```

```
generating makefile
makefile done
Build Project - project1
COBOL-IT Environement set to /opt/cobol-it-64
building /root/data/devstudio/CDS2.0/workspace/project1/newcall.cbl
cobc newcall.cbl -fixed -fsign-ascii -m -o "./newcall.so" -MT "./newcall.so" -MF .cobolit.tmp.172.17.0.2/newcall-3decdd18e4fbb80b1c983df41158d5a0b6803e84.d
COBOL-IT Build complete.
```

Image 13- Two Debugging Sessions

## The Debugger Window- Focus on C Debugger Thread 1

Within the “C” debugger, there are 2 threads. Thread 2 is the COBOL Debugger Runtime. Thread 1 is the program running. Select Thread 1 by clicking on it.
Now, you can open c_printf and see the “C” Debugger.


COBOL-IT Debugger - Test3Co
File Edit Source Navigate Search Project Run Window Help
COBOL-IT Developer Studio
COBOL-IT Debugger
Quick Access
Debug
Cobol Program
Thread[1]
NEWCALL () line: 16
sh (Jan 8, 2015 5:09:29 PM)
Test3Cobol [C/C++ Attach to Application]
newcall [18903] [cores: 1]
Thread [2] 18904 [core: 1] (Suspended : Container)
Thread [1] 18903 [core: 1] (Suspended : User Request)
gdb
newcall.cbl
nanosleep() at
c_printf.c
debug.c
common.c
cobcdb.c
Console
Tasks
Problems
Executables

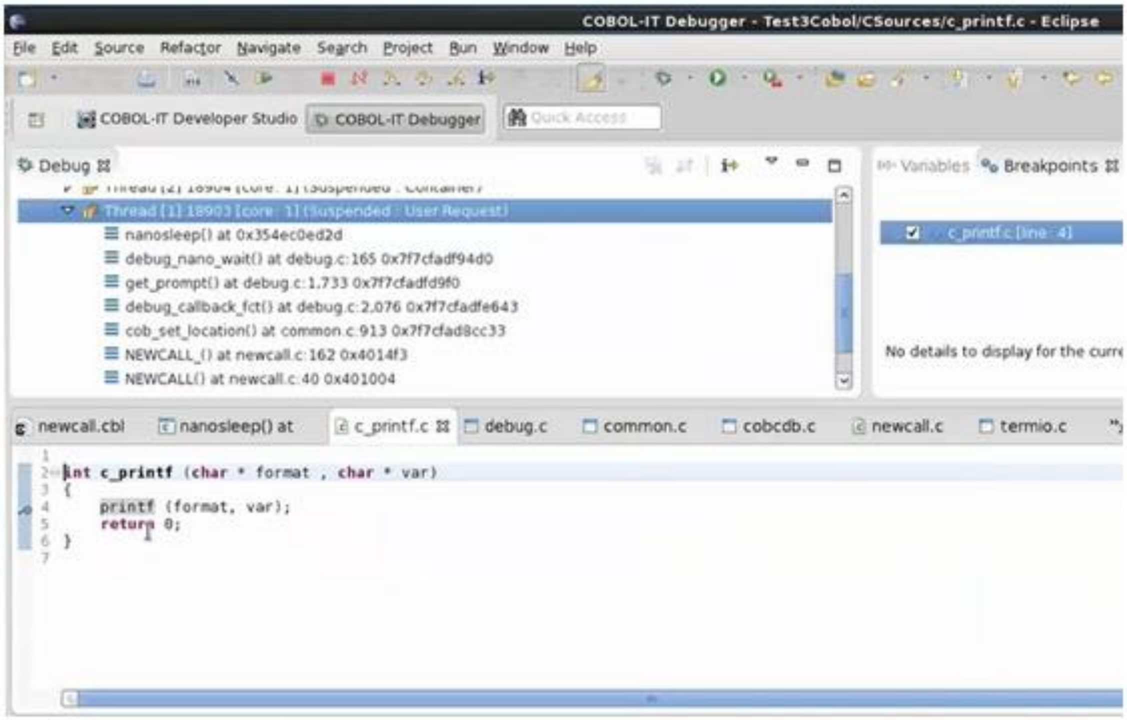
```
       OBJECT-COMPUTER. NCR.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 MYDATA PIC X(9).
       01 MYLOC PIC X(9) VALUE "coco".
       PROCEDURE DIVISION .
       START-NEWSUB.
       DISPLAY "NEWCALL Stared"
       CALL "C$DEBUG" USING 12345

       CALL "NEWSUB" using "A" "B"
       CALL "c_printf" USING "%s" "Hello"
       EXIT PROGRAM.
```

```
[Debug Console] .hello (2) [Cobol Program] sh (Jan 8, 2015 5:09:29 PM)
COBOL-IT Environement set to /opt/cobol-it-debug-64
Start debugging Poccess ID 18903
             Debugger ID 12345
```

Image 14- Focus on the “C” Debugger

## The “C” Debugger- Set a Breakpoint

Set a breakpoint in c_printf.c.
The breakpoint will be displayed in the breakpoint window.
Return to newcall.cbl, and we will enter the “C” function, using the Step Into ( F5) function.


COBOL-IT Debugger - Test3Cobol/CSources/c_printf.c - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
COBOL-IT Developer Studio
COBOL-IT Debugger
Quick Access
Debug
Variables
Breakpoints
newcall.cbl
nanosleep() at
c_printf.c
debug.c
common.c
cobcdb.c
newcall.c
termio.c

Thread [2] 18904 [core: 1] (Suspended : Container)
Thread [1] 18903 [core: 1] (Suspended : User Request)
nanosleep() at 0x354ec0ed2d
debug_nano_wait() at debug.c:165 0x7f7cfadf94d0
get_prompt() at debug.c:1,733 0x7f7cfadfd9f0
debug_callback_fct() at debug.c:2,076 0x7f7cfadfe643
cob_set_location() at common.c:913 0x7f7cfad8cc33
NEWCALL_() at newcall.c:162 0x4014f3
NEWCALL() at newcall.c:40 0x401004

☑ c_printf.c [line: 4]

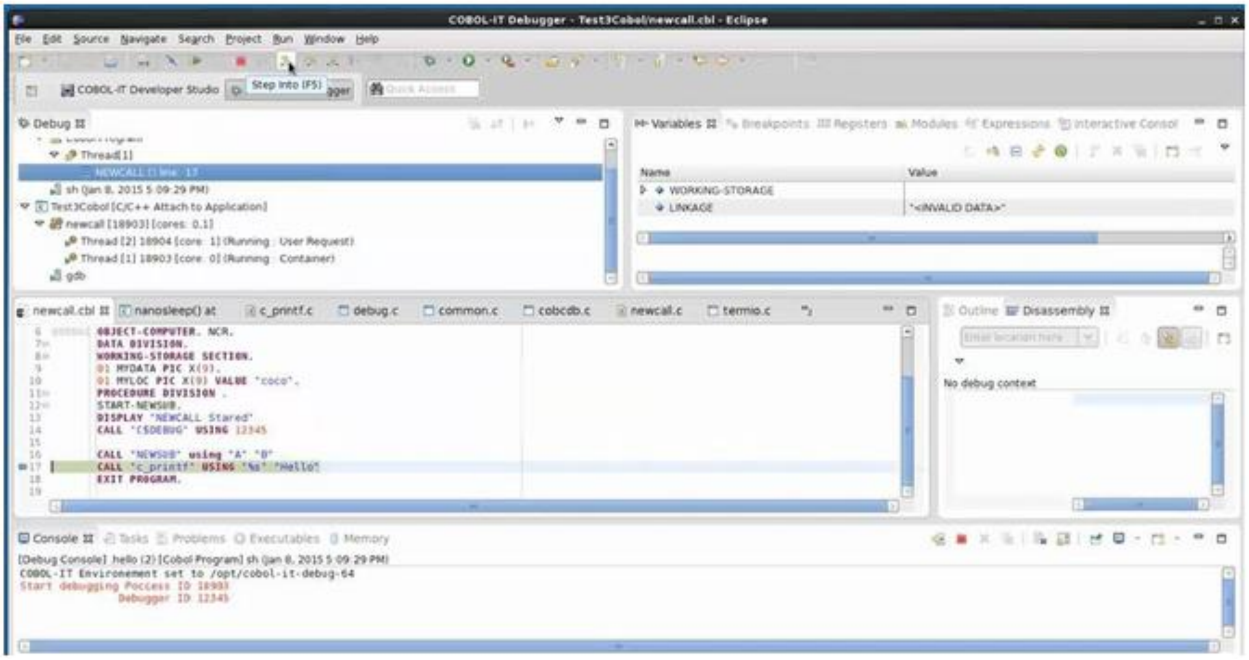
No details to display for the curr

```
int c_printf (char * format , char * var)
{
    printf (format, var);
    return 0;
}
```

Image 15- Set a breakpoint in the “C” compiler

## Step Into the C Function (F5)

With the Cursor on the CALL “c_printf” statement, use the Step Into (F5) function to step into the “C” program.
Open c_printf.c to debug.



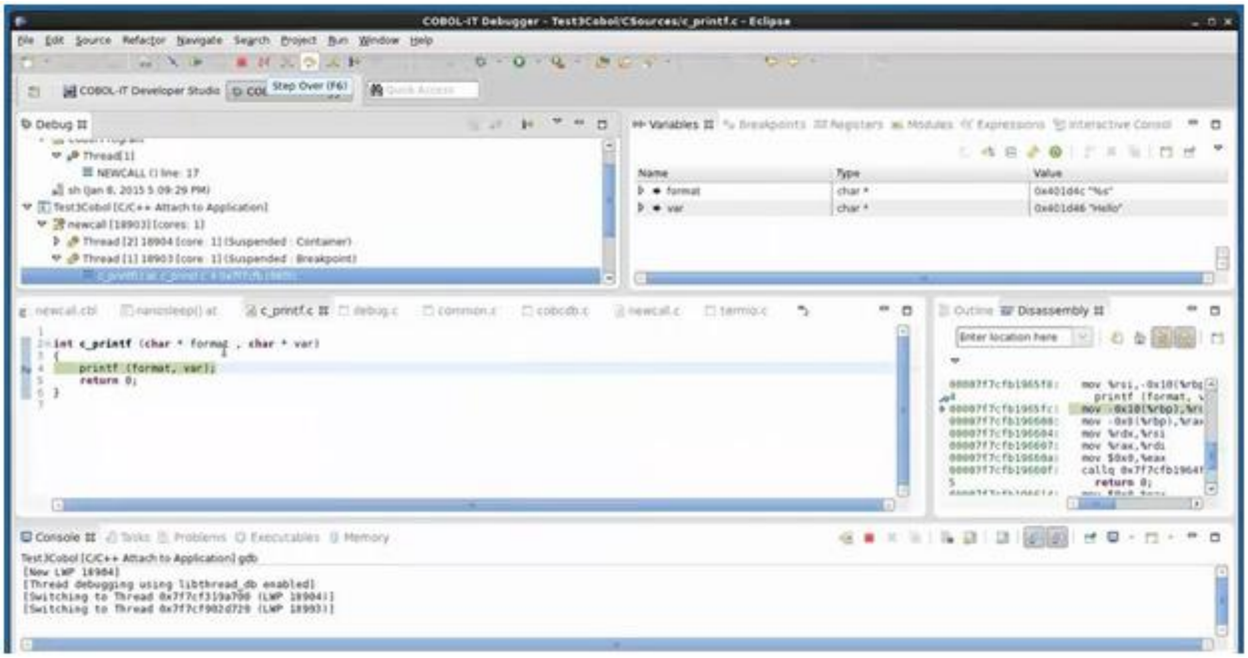
```
       OBJECT-COMPUTER. NCR.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 MYDATA PIC X(9).
       01 MYLOC PIC X(9) VALUE "coco".
       PROCEDURE DIVISION .
       START-NEWSUB.
       DISPLAY "NEWCALL Stared"
       CALL "CSDEBUG" USING 12345

       CALL "NEWSUB" using "A" "B"
       CALL "c_printf" USING "%s" "Hello"
       EXIT PROGRAM.
```

```
[Debug Console] hello (2) [Cobol Program] sh (Jan 8, 2015 5:09:29 PM)
COBOL-IT Environement set to /opt/cobol-it-debug-64
Start debugging Poccess ID 18903
                Debugger ID 12345
```

Image 16- Step into the “C” debugger

## The “C” debugger

Now, you can debug on the “C” side. Variable values are in the Variable window. You can use the Step Functions.
The “C” debugger will stop on breakpoints.
Use the Continue function to return to the COBOL program.  Set a breakpoint in the COBOL program and resume using the COBOL debugger.



```
Thread[1]
    NEWCALL () line: 17
sh (Jan 8, 2015 5:09:29 PM)
Test3Cobol [C/C++ Attach to Application]
  newcall [18903] [cores: 1]
    Thread [2] 18904 [core: 1] (Suspended : Container)
    Thread [1] 18903 [core: 1] (Suspended : Breakpoint)
```

| Name | Type | Value |
|---|---|---|
| format | char * | 0x401d4c "%s" |
| var | char * | 0x401d46 "Hello" |

```
int c_printf (char * format , char * var)
{
    printf (format, var);
    return 0;
}
```

```
00007f7cfb1965f8:   mov %rsi,-0x10(%rbp
                      printf (format, v
00007f7cfb1965fc:   mov -0x10(%rbp),%r
00007f7cfb196600:   mov -0x8(%rbp),%ra
00007f7cfb196604:   mov %rdx,%rsi
00007f7cfb196607:   mov %rax,%rdi
00007f7cfb19660a:   mov $0x0,%eax
00007f7cfb19660f:   callq 0x7f7cfb1964f
5                       return 0;
```

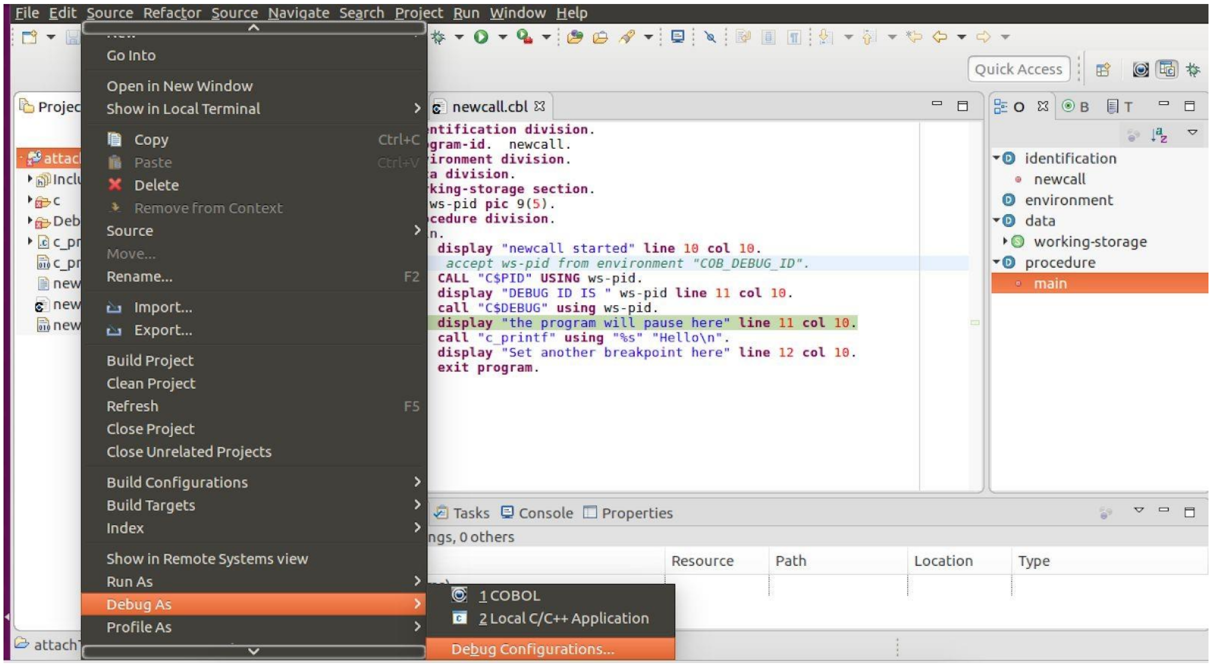
```
Test3Cobol [C/C++ Attach to Application] gdb
[New LWP 18904]
[Thread debugging using libthread_db enabled]
[Switching to Thread 0x7f7cf319a700 (LWP 18904)]
[Switching to Thread 0x7f7cf902d720 (LWP 18903)]
```

[Image 17]- Stopping on a breakpoint in the “C” Debugger

# Cobol and C/C++ debugging

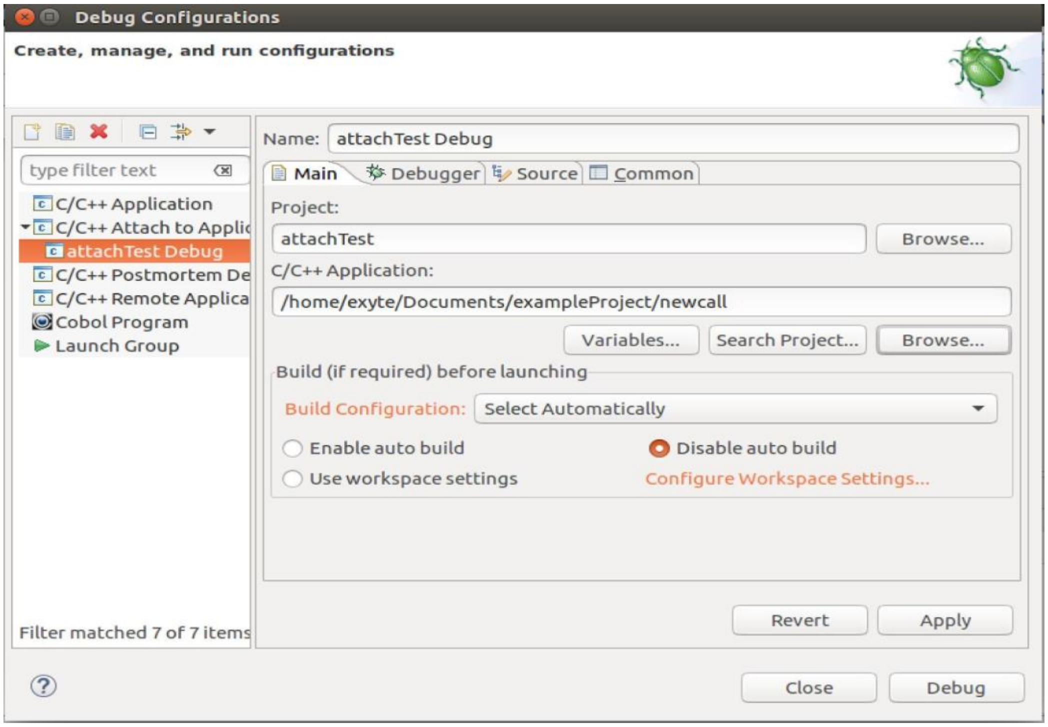
yyyyy

1. Open project debug configurations:



```
ntification division.
gram-id.  newcall.
ironment division.
a division.
king-storage section.
ws-pid pic 9(5).
cedure division.
n.
 display "newcall started" line 10 col 10.
  accept ws-pid from environment "COB_DEBUG_ID".
 CALL "C$PID" USING ws-pid.
 display "DEBUG ID IS " ws-pid line 11 col 10.
 call "C$DEBUG" using ws-pid.
 display "the program will pause here" line 11 col 10.
 call "c_printf" using "%s" "Hello\n".
 display "Set another breakpoint here" line 12 col 10.
 exit program.
```

2. Configure C/C++ debug attach as shown below, note that only Main tab is important. Press apply and debug.



Debug Configurations

Create, manage, and run configurations

Name: attachTest Debug

Project: attachTest

C/C++ Application: /home/exyte/Documents/exampleProject/newcall

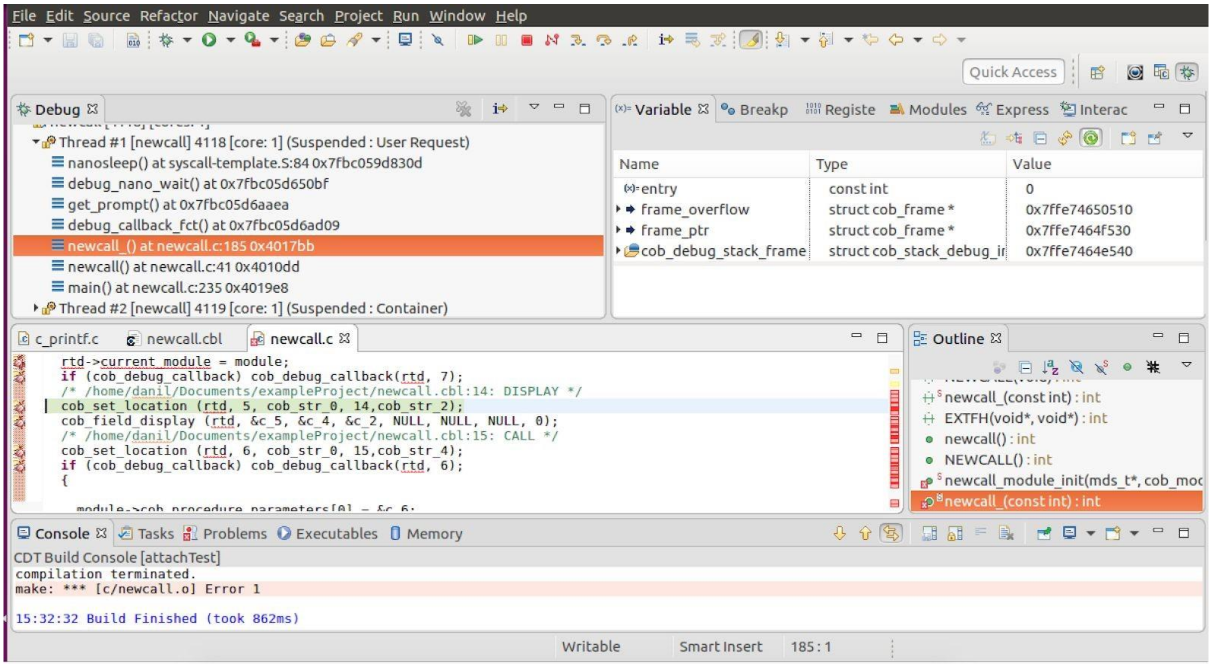
Build (if required) before launching

Build Configuration: Select Automatically

○ Enable auto build ● Disable auto build

○ Use workspace settings

3. Now under Debug perspective in second debug stack frame you can find C source files from “c” folder which were configured on step 6.



Thread #1 [newcall] 4118 [core: 1] (Suspended : User Request)
nanosleep() at syscall-template.S:84 0x7fbc059d830d
debug_nano_wait() at 0x7fbc05d650bf
get_prompt() at 0x7fbc05d6aaea
debug_callback_fct() at 0x7fbc05d6ad09
newcall_() at newcall.c:185 0x4017bb
newcall() at newcall.c:41 0x4010dd
main() at newcall.c:235 0x4019e8
Thread #2 [newcall] 4119 [core: 1] (Suspended : Container)

| Name | Type | Value |
|---|---|---|
| entry | const int | 0 |
| frame_overflow | struct cob_frame * | 0x7ffe74650510 |
| frame_ptr | struct cob_frame * | 0x7ffe7464f530 |
| cob_debug_stack_frame | struct cob_stack_debug_ir | 0x7ffe7464e540 |

c_printf.c newcall.cbl newcall.c

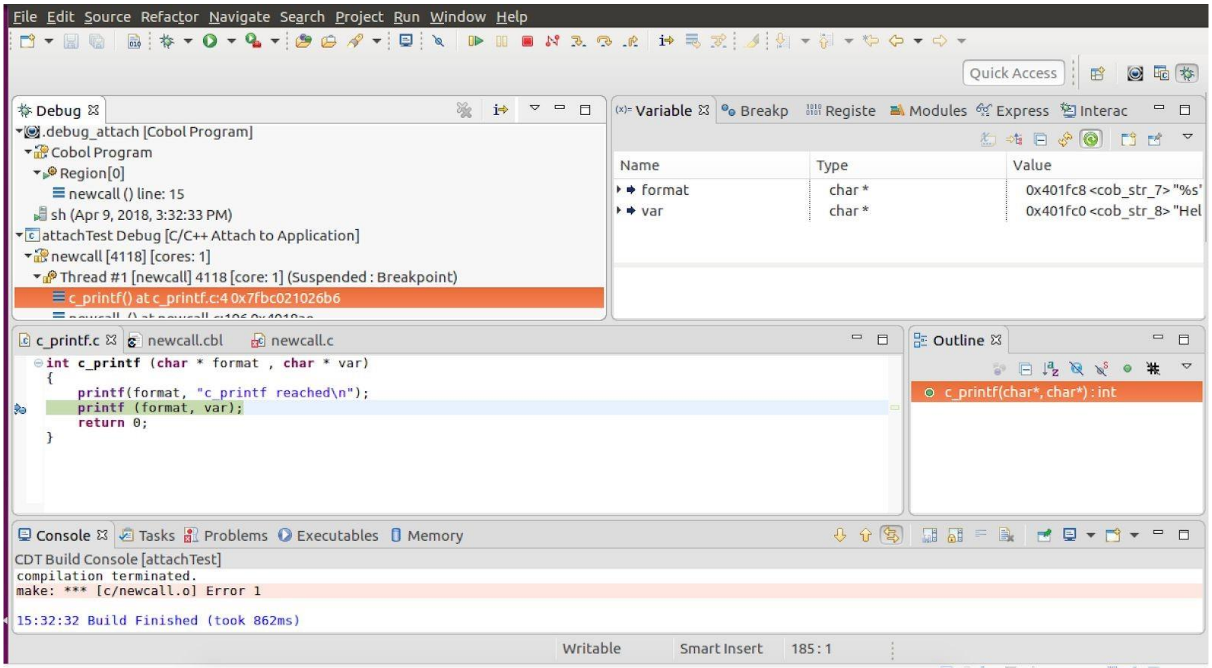
```
    rtd->current_module = module;
    if (cob_debug_callback) cob_debug_callback(rtd, 7);
    /* /home/danil/Documents/exampleProject/newcall.cbl:14: DISPLAY */
    cob_set_location (rtd, 5, cob_str_0, 14,cob_str_2);
    cob_field_display (rtd, &c_5, &c_4, &c_2, NULL, NULL, NULL, 0);
    /* /home/danil/Documents/exampleProject/newcall.cbl:15: CALL */
    cob_set_location (rtd, 6, cob_str_0, 15,cob_str_4);
    if (cob_debug_callback) cob_debug_callback(rtd, 6);
    {

      module->cob_procedure_parameters[0] = &c_6;
```

newcall_(const int) : int
EXTFH(void*, void*) : int
newcall() : int
NEWCALL() : int
newcall_module_init(mds_t*, cob_moc
newcall_(const int) : int

CDT Build Console [attachTest]

```
compilation terminated.
make: *** [c/newcall.o] Error 1

15:32:32 Build Finished (took 862ms)
```

4. Now there are two running debuggers. You should click "Resume" in both debuggers and you will see that debugger will stop on breakpoint in C program.



.debug_attach [Cobol Program]
Cobol Program
Region[0]
newcall () line: 15
sh (Apr 9, 2018, 3:32:33 PM)
attachTest Debug [C/C++ Attach to Application]
newcall [4118] [cores: 1]
Thread #1 [newcall] 4118 [core: 1] (Suspended : Breakpoint)
c_printf() at c_printf.c:4 0x7fbc021026b6

| Name | Type | Value |
|---|---|---|
| format | char * | 0x401fc8 <cob_str_7> "%s' |
| var | char * | 0x401fc0 <cob_str_8> "Hel |

c_printf.c newcall.cbl newcall.c

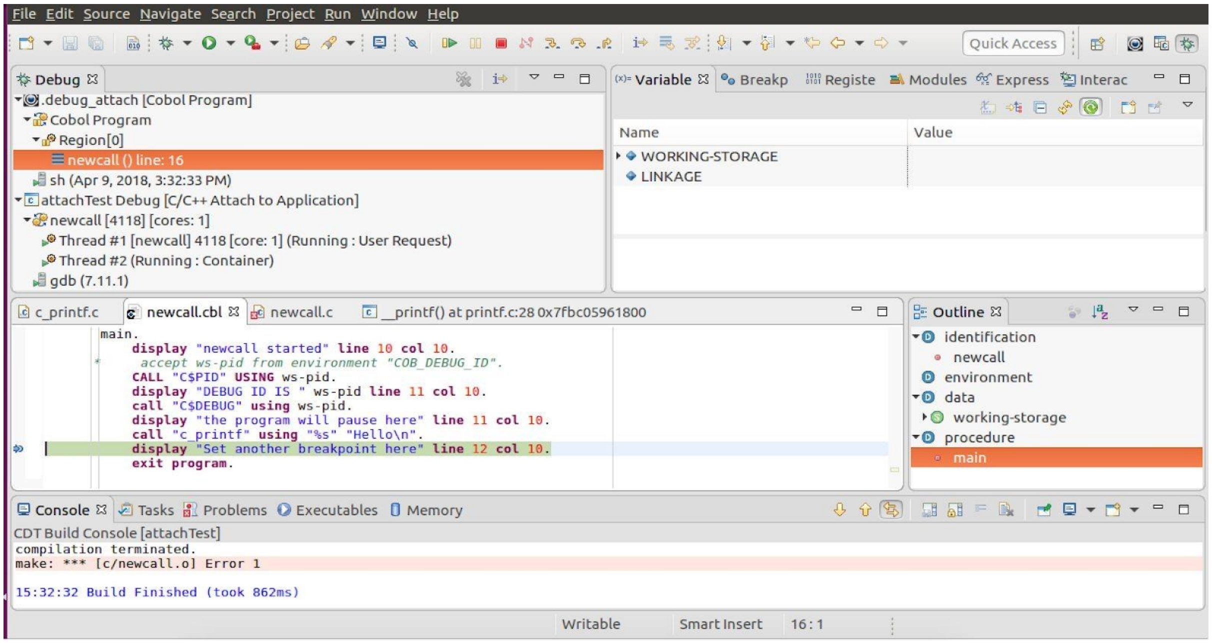
```
int c_printf (char * format , char * var)
{
    printf(format, "c_printf reached\n");
    printf (format, var);
    return 0;
}
```

c_printf(char*, char*) : int

CDT Build Console [attachTest]

```
compilation terminated.
make: *** [c/newcall.o] Error 1

15:32:32 Build Finished (took 862ms)
```

5. Now you can debug C program and when you need to switch to a breakpoint in Cobol source file you can just click "Resume" in C debugger and debugger will stop on next Cobol breakpoint.



```
.debug_attach [Cobol Program]
  Cobol Program
    Region[0]
      newcall () line: 16
  sh (Apr 9, 2018, 3:32:33 PM)
attachTest Debug [C/C++ Attach to Application]
  newcall [4118] [cores: 1]
    Thread #1 [newcall] 4118 [core: 1] (Running : User Request)
    Thread #2 (Running : Container)
  gdb (7.11.1)
```

```
       main.
           display "newcall started" line 10 col 10.
      *     accept ws-pid from environment "COB_DEBUG_ID".
           CALL "C$PID" USING ws-pid.
           display "DEBUG ID IS " ws-pid line 11 col 10.
           call "C$DEBUG" using ws-pid.
           display "the program will pause here" line 11 col 10.
           call "c_printf" using "%s" "Hello\n".
           display "Set another breakpoint here" line 12 col 10.
           exit program.
```

```
CDT Build Console [attachTest]
compilation terminated.
make: *** [c/newcall.o] Error 1

15:32:32 Build Finished (took 862ms)
```

## In parting…

C/C++ Attach to Application is included with the Eclipse IDE for C/C++ Developers, and required.
Gdb cannot be run as a remote debugger. Both your main COBOL program and "C" program must be compiled with the –G compiler flag.

### *Attaching to the Debugger when calling COBOL from "C"*

***C$DEBUG is a COBOL routine, so it can only be called from within a COBOL program. In a CICS environment, what is needed is a mechanism to set up the reverse attach debugger connection before the program is called. Can COBOL-IT provide an API function that can be called before the program is called, so that when the runtime is started from « C », the runtime can set the breakpoint at the start of the program, and attach to the debugger with the specified debug-id ?***

Yes- Before stating the entry COBOL program, call :

```
cob_runtime_debugger_activate(rtd, did);
```

Where *did* is the debug ID used in eclipse to connect to the debugger.  The runtime will then stop on the next COBOL statement.

**www.cobol-it.com**

June, 2020