COBOL-IT Developer Studio
The Debugger Perspective
Getting Started
Version 1.8
COBOL-IT DEVELOPER STUDIO TOPICS

Introduction ......................................................................................................................... 4

COBOL-IT Developer Studio License terms ........................................................................ 4

DEPENDENCIES .................................................................................................................. 5

The COBOL-IT Developer Studio Distribution .................................................................. 5

THE DEBUGGER PERSPECTIVE ......................................................................................... 7

Configuration of the Debugger Perspective ......................................................................... 7

Window>Preferences>Run/Debug>Perspectives .................................................................... 7

Window>Preferences>General>Keys .................................................................................... 8

Window>Show View ........................................................................................................ 9

Important Debugging Operations ...................................................................................... 10

Opening the Debugger Perspective ................................................................................... 11

The Debug View ............................................................................................................. 15

Terminating a Debug Session- Key concepts .................................................................... 16

Step/Run/Terminate Functions ......................................................................................... 17

The Variables View ........................................................................................................ 19

Breakpoints ...................................................................................................................... 23

Expressions ....................................................................................................................... 23

Code Editor Functions ..................................................................................................... 24

COBOL-IT Debugger Reference ....................................................................................... 25

The Views in the COBOL-IT Debugger Perspective .......................................................... 25

The Debug View ............................................................................................................. 25

The COBOL-IT Program View ........................................................................................ 27

The Variable View ........................................................................................................... 28

The Breakpoints View ..................................................................................................... 29

The Expressions View ..................................................................................................... 30

Qualified names in the Expression View ........................................................................... 31

The Outline View ............................................................................................................ 32

The Console View ............................................................................................................ 32

The Tasks View ................................................................................................................. 33

The Problems View ......................................................................................................... 34
Attaching the Debugger to a Running Process...............................................................................34
  Launch, and Pause the Runtime using CALL « CSDEBUG »......................................................35
  Attach the Debugger from the Developer Studio.......................................................................36
  Create a New COBOL Project......................................................................................................36
  Select Debug Attach Function......................................................................................................37
  Select ID ......................................................................................................................................37
  Set Environment Variables ..........................................................................................................38
  Edit Source Lookup Path ............................................................................................................39
  The Source Not Found message ..................................................................................................39
  Debug in the Developer Studio....................................................................................................40
  Programs used in this sample........................................................................................................41

Attaching a “C” debugger to a running process............................................................................45
  -G.................................................................................................................................................45
  -fnostrip......................................................................................................................................45
  Overview.......................................................................................................................................46
  Pre-requisites ................................................................................................................................46
  Sample Programs ............................................................................................................................46
  Compile and link the sample programs..........................................................................................47
  Create a New COBOL project........................................................................................................48
  Launch the newcall executable ....................................................................................................48
  The Debug Attach function ..........................................................................................................49
  The Reverse Attach Window .........................................................................................................50
  The COBOL-IT Debugger Attach ..................................................................................................50
  C/C++ Debug Configuration .........................................................................................................51
  Create a new C/C++ Attach to Application ...................................................................................52
  C/C++ Debug Configuration- Main Tab..........................................................................................53
  C/C++ Debug Configuration- Debugger Tab ..................................................................................54
  C/C++ Debug Configuration- Source Tab.....................................................................................55
  C/C++ Debug Configuration- Common Tab...................................................................................56
  Select newcall from list of running applications ..........................................................................57
  The Debugger Window - Two Debugging Sessions .......................................................................58
  The Debugger Window- Focus on C Debugger Thread 1...............................................................59
  The “C” Debugger- Set a Breakpoint.............................................................................................60
  Step Into the C Function (F5).........................................................................................................61
  The “C” debugger .........................................................................................................................62
  In parting........................................................................................................................................62
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

The copyright for the COBOL-IT Developer Studio® is wholly owned by COBOL-IT. Any unauthorized reproduction of the software without the express written consent of COBOLIT is prohibited.

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

COBOL-IT, COBOL-IT Compiler Suite, CitSQL, CitSORT, CitXML, and COBOL-IT Developer Studio are trademarks or registered trademarks of COBOL-IT.
Eclipse is a trademark of the Eclipse Foundation.
IBM, and AIX are registered trademarks of International Business Machines Corporation.
Linux is a registered trademark of Linus Torvalds.
Windows, Visual Studio, and Visual Studio Express are registered trademarks of Microsoft Corporation.
Java and Solaris are registered trademarks of Sun Microsystems, Inc.
UNIX is a registered trademark of The Open Group
HP is a registered trademark of Hewlett Packard, Inc.
Red Hat is a registered trademark of Red Hat, Inc.
SUSE is a registered trademark of Novell, Inc.
All other trademarks are the property of their respective owners.
## Dependencies

Dependencies for the Developer Studio are:

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>“C” compiler</td>
<td>The COBOL-IT Compiler requires a “C” compiler. While most Linux&gt;Unix installations will include a “C” compiler, many Windows installations will not. Windows users can download the “express” edition of the Visual Studio 2008 C++ 32-bit compiler at: http://&gt;www.microsoft.com&gt;express&gt;downloads&gt;</td>
</tr>
<tr>
<td>COBOL-IT Compiler Suite</td>
<td>When accessing projects located on the Windows platform, the Compiler license for the Windows platform is required. When accessing projects located on a remote platform, the Compiler license for the remote platform is required. In a configuration in which projects are accessed locally and remotely, Compiler licenses for both the local and remote hosts are required. The COBOL-IT Compiler Suite, Enterprise Edition can be downloaded from COBOL-IT’s online portal. For access to COBOL-IT’s online portal, please contact your sales representative at <a href="mailto:sales@cobol-it.com">sales@cobol-it.com</a>. The COBOL-IT Compiler Suite setup file is run during the startup of the Developer Studio, so the Compiler Suite must be installed.</td>
</tr>
<tr>
<td>Java Runtime Environment (JRE)</td>
<td>The COBOL-IT Developer Studio can be run with the Java Runtime Environment (JRE) Version 1.6 or greater. The JRE can be downloaded at: http://&gt;java.sun.com&gt;javase&gt;downloads&gt;index.jsp</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Eclipse is included with the download of Developer Studio.</td>
</tr>
</tbody>
</table>

The COBOL-IT Developer Studio requires that the COBOL-IT Compiler Suite already be installed on the host platform, and that a “C” compiler be installed on the host platform.

The COBOL-IT Developer Studio is an Eclipse plug-in, and as such, requires that Eclipse be installed on the host platform. Eclipse, in turn, requires that a Java Runtime Environment (JRE) be installed on the host platform.

## The COBOL-IT Developer Studio Distribution

The COBOL-IT Developer Studio, Enterprise Edition can be downloaded at www.cobol-it-
online.com, with a login and password provided by your sales representative. To request an evaluation copy of the COBOL-IT Developer Studio, Enterprise Edition, please contact your sales representative at sales@cobol-it.com.

The COBOL-IT Developer Studio, Enterprise Edition is available with Subscription. The COBOL-IT Developer Studio, Enterprise Edition provides the following functionality:

- Allows use of the COBOL-IT Compiler Suite Enterprise Edition
- Provides Debugger Perspective both locally, and on Remote Systems
- Provides a Remote Systems Perspective, allowing use of Compiler, Runtime, and Debugger functionalities on remote machines.
The COBOL-IT Debugger operates inside the COBOL-IT Debugger perspective. When the COBOL-IT Debugger perspective is active, you will see the COBOL-IT Debugger Perspective push-button in the upper right-hand corner of your screen.

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

<table>
<thead>
<tr>
<th>Debugger Function</th>
<th>Hot Key</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Step Into" /></td>
<td>F5</td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Step Over" /></td>
<td>F6</td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Step Return" /></td>
<td>F7</td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Resume" /></td>
<td>F8</td>
<td>Resumes the execution of the program.</td>
</tr>
<tr>
<td><img src="image" alt="Debug" /></td>
<td>F11</td>
<td>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.</td>
</tr>
<tr>
<td><img src="image" alt="Terminate" /></td>
<td>Ctrl+F2</td>
<td>Terminates the program being debugged.</td>
</tr>
<tr>
<td><img src="image" alt="Run" /></td>
<td>Ctrl+F11</td>
<td>Runs the program, skipping all breakpoints.</td>
</tr>
<tr>
<td><img src="image" alt="Toggle Breakpoint" /></td>
<td>Ctrl+Shift+B</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Window>Show View**

To Open, or Show a View, Click on Window>Show View, and then select a view that you wish to Show from the dropdown list.

The Views that are displayed in the COBOL-IT Debugger Perspective can be removed, or “Closed” in Eclipse terminology, by clicking on them, and then clicking on the “X” in the right-hand part of the Tab title. Views that have been closed are not visible.

For more information, see **Using “Views” in Eclipse**.
Important Debugging Operations
Opening the Debugger Perspective

Open Perspectives Tab

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.

Opening the Debugger Perspective to Debug a Specific Program

When the Workspace has been configured, as described above, to always “Open the associated perspective when launching, then there are several ways to launch the Debugger perspective for the purpose of debugging a specific program.

Press the Debug button, or F11

Select a source file that has been set up in the project, and press the Debug toolbar
button, or 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 effect of pressing the Debug button will depend on other Workspace configurations. COBOL-IT recommends Developer Studio configuration settings of Window>Preferences>General>Workspace>Save Automatically before Build. Getting Started with COBOL-IT also stresses the advantages of performing Builds within a Clean operation, prior to performing a Debug or Run operation, though this is not required.

The Debug / F11 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 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.

We name our new Debug Configuration “hello.dbg” and associate our new Debug Configuration with the program “hello.cbl”.
COBOL-IT Developer Studio– Getting Started
The Debugger Perspective

Version 1.8
After selecting your Debug Configuration, click on the “Debug” button to launch the Debugger Perspective.
The Debug View

In the image below:

- hello.dbg is the name of the debug configuration
- COBOL Program is the class of program running
- Thread(1) is the first thread running in the Program
- hello () line 8 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.
Terminating a Debug Session - Key concepts

- The dated, time-stamped entry represents the shell script that is actually running the debug session. When focus is on the shell script, only the Terminate function is enabled.
- The termination of the Debug Session corresponds to the termination of this shell script.
- To terminate the shell script, and properly terminate the debug session, click on the red terminate button on the Debug View toolbar.

- When terminated, the debug script is marked “terminated” with an exit value.
- If you do not properly terminate a debug session, Eclipse will return an error.
- Caution- Stepping through a “Stop Run” statement does not properly terminate a debug session. You must click on the red terminate button on the Debug View toolbar.
In our subsequent samples, we will reference the programs resmain.cbl and rescheck.cbl that are also included in our project.

**Step/Run/Terminate Functions**

**Single-Step**

The image below is designed to demonstrate the progression of four Step Into (F5) commands.

**Step Over**

The image below is designed to demonstrate the progression of a Step Into (F5) / Step Over (F6) / Step Into (F5) sequence of commands. When the cursor is positioned on a PERFORM or CALL statement, the Step Over (F6) command causes all of the code in the target of the PERFORM/CALL to be executed, and then steps ahead to the next line of code. This action of the Step Over (F6) command will pause on Format 1/Format 2 ACCEPT Statements, and will pause on breakpoints.
Step Return

The image below is designed to demonstrate the progression of a Step Into (F5) / Step Into (F5) / Step Return (F7) sequence of commands. When the cursor has entered the target of a PERFORM or CALL statement, the Step Return (F7) command causes the remaining code in the target of the PERFORM/CALL to be executed, and then steps ahead to the next line of code. This action of the Step Return (F7) command will pause on Format 1/Format 2 ACCEPT Statements, and will pause on breakpoints.

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.
The Terminate command terminates the program at the current line of execution.

By default, the COBOL-IT Debugger Perspective is configured to Auto-Refresh the Variable view.
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… The Change Value 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.
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. When setting Hexadecimal values, note that invalid Hexadecimal values are not permitted.

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.

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

### 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. Note that when adding a variable as an Expression, the variable-name must be entered inside quotations. Thus, to enter the variable reslist-stat as a Watched variable, you would enter the name as “reslist-stat” (including quotation marks).
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.

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:

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:

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

The Debug View

The Debug View shows a stack view of the execution of your program. In the graphic below, you can see the program is currently executing at line 52 in the program rescheck, and that rescheck was called from resmain on line 137.

The Debug view toolbar

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove all terminated launches</td>
<td>Clears terminated launches from the Debug View</td>
</tr>
<tr>
<td>Connect to a process</td>
<td>Initiates a Remote Debugging session.</td>
</tr>
<tr>
<td>Resume (F8)</td>
<td>Resumes the execution of the program.</td>
</tr>
<tr>
<td>Suspend</td>
<td>Suspends execution of the program, and re-enters the debugger.</td>
</tr>
<tr>
<td>Terminate (Ctrl+F2)</td>
<td>Terminates the currently running process.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Terminates a Remote Debugging session.</td>
</tr>
<tr>
<td>Step Into (F5)</td>
<td>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.</td>
</tr>
<tr>
<td>Step Over (F6)</td>
<td>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.</td>
</tr>
<tr>
<td>Step Return (F7)</td>
<td>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.</td>
</tr>
<tr>
<td>Drop to Frame</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Instruction Stepping Mode</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Use Step Filters</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>View Menu</td>
<td>Drops down a view of Menu options</td>
</tr>
</tbody>
</table>
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 56. This marks the line of code that the debugger will execute next. Note also the circle positioned on line 60. This marks a breakpoint, that is set within the source code.

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 a “Change Value” function, which allows the displayed Value to be changed. Variables can be selected, and copied to the clipboard. A Find function allows for the searching of a variable name.

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Type Names</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Show Logical Structure</td>
<td>COBOL data items are all presented in their logical structures, and can be expanded and collapsed.</td>
</tr>
<tr>
<td>Collapse All (Ctrl+Shift+Numpad-)</td>
<td>Collapses all structures back to their parent</td>
</tr>
<tr>
<td>Divide</td>
<td>Sections (Working-Storage, File, Linkage)</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Auto Refresh</td>
<td>Auto-Refresh function. Working Storage, File Section, and Linkage Section are auto-refreshed in real time, while debugging the program.</td>
</tr>
<tr>
<td>Add Global Variables</td>
<td>Not Supported in the Debugger Perspective</td>
</tr>
<tr>
<td>Remove Selected Global Variables</td>
<td>Not Supported in the Debugger Perspective</td>
</tr>
<tr>
<td>Remove All Global Variables</td>
<td>Not Supported in the Debugger Perspective</td>
</tr>
<tr>
<td>View Menu</td>
<td>Drops down a view of Menu options</td>
</tr>
</tbody>
</table>

### 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.

### Hit Count

### 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

Remove all 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.
Qualified names in the Expression View

The debugger supports qualified names, such as data-element-1 in group-2 in the Expression View.

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Type Names</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Show Logical Structure</td>
<td>COBOL data items are all presented in their logical structures, and can be expanded and collapsed.</td>
</tr>
<tr>
<td>Collapse All (Ctrl+Shift+Numpad-Divide)</td>
<td>Collapses all structures .</td>
</tr>
<tr>
<td>Create a new Watch Expression</td>
<td>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.</td>
</tr>
<tr>
<td>Remove Selected Expressions</td>
<td>Removes expressions that have been selected by clicking on them.</td>
</tr>
<tr>
<td>Remove All Expressions</td>
<td>Removes all expressions listed in the Expressions View.</td>
</tr>
<tr>
<td>View Menu</td>
<td>Drops down a view of Menu options</td>
</tr>
</tbody>
</table>
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 View Example]

The Outline View Toolbar

The Outline View Toolbar contains an Alphabetic Sort function.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Focus" /></td>
<td>Focus on Active Task</td>
</tr>
<tr>
<td><img src="image" alt="Sort" /></td>
<td>Sort</td>
</tr>
<tr>
<td><img src="image" alt="View Menu" /></td>
<td>View Menu</td>
</tr>
</tbody>
</table>

The Console View

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

<table>
<thead>
<tr>
<th>Toolbar Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Console</td>
<td>Clears all text from the open console window.</td>
</tr>
<tr>
<td>Scroll Lock</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Pin Console</td>
<td>Not Supported in the COBOL-IT Debugger Perspective</td>
</tr>
<tr>
<td>Display Selected Console</td>
<td>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.</td>
</tr>
<tr>
<td>Open Console</td>
<td>Adds a new console to the Display Selected Console dropdown list of available consoles.</td>
</tr>
</tbody>
</table>

**The Tasks View**

**The Tasks View Toolbar**

<table>
<thead>
<tr>
<th>Toolbar Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on Active Task</td>
<td>Places focus in active source file.</td>
</tr>
<tr>
<td>View Menu</td>
<td>Drops down a view of Menu options</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Icon</th>
<th>Focus on Active Task</th>
<th>Places focus in active source file.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>View Menu</td>
<td>Drops down a view of Menu options</td>
</tr>
</tbody>
</table>

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 launching (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.
Attach the Debugger from the Developer Studio

The Developer Studio must be configured to enter the Debugger Perspective when launching the debugger.

Create a New COBOL Project
In the Navigator Window, right-click on the Project, and select COBOL>Debug Attach from the dropdown list:

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.
## Set Environment Variables

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

### 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 source code of subpgm.cbl with the project, click on the Edit Source Lookup Path… button.
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.

After clicking the Add button, select File System Directory, Click Ok.
Use the Browse button to locate the Source Location

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

You are now ready to debug in the Developer Studio:

Use the Debugger toolbar buttons to debug your program.

Terminate the Debugger by clicking on the Terminate button.
Programs used in this sample

debgid.cbl

IDENTIFICATION DIVISION.
  PROGRAM-ID. debugid.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
PROCEDURE DIVISION.
MAIN.
  DISPLAY "C$DEBUG Usage", LINE 6 COL 10.
  CALL "subpgm".
  DISPLAY "Back in DebugID..." LINE 20 COL 10.
STOP RUN.

subpgm.cbl

IDENTIFICATION DIVISION.
  PROGRAM-ID. subpgm.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
  77 ws-process-id PIC 9(5).
PROCEDURE DIVISION.
MAIN.
    CALL "C$PID" USING ws-process-id.
    DISPLAY "PAUSING IN SUBPGM" LINE 12 COL 10.
    CALL "C$DEBUG" USING ws-process-id.
    DISPLAY "YOU CAN DEBUG NOW" LINE 16 COL 10.
    DISPLAY "AND" LINE 17 COL 10.
    DISPLAY "FIND THE PROBLEM" LINE 18 COL 10.
    EXIT PROGRAM.

runit.bat

SET COB_LIBRARY_PATH=\object
cobc -o \object -g debugid.cbl
cobc -o \object -g subpgm.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 debugging 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.
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. 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.

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

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.

The following must be installed on your Linux machine:

- Eclipse IDE for C/C++ Developers
- The gdb source level debugger
- COBOL-IT Compiler Suite Enterprise Edition
- COBOL-IT Developer Studio

The Debugger Perspective

Pre-requisites

The following must be installed on your Linux machine:

- Eclipse IDE for C/C++ Developers
- The gdb source level debugger
- COBOL-IT Compiler Suite Enterprise Edition
- COBOL-IT Developer Studio

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.

```c
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 –b –o clibs.so –v The library, clibs.so, is compiled with debugging information.

The above command is the equivalent of using :
>gcc -g

Compile newcall.cbl with COBOL debugging information (-g) and C debugging information (-G).
Link clibs.so, and create an executable (-x) called newcall (-o).
The -v compiler flag provides details about the compile.
> cocab newcall.cbl -g -x clibs.so -G -v -o newcall  
This creates an executable module called newcall, in which clibs.so is linked to the main module.

**Create a New COBOL project**

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

![Image 1- The COBOL Project, with files](image1)

**Launch the newcall executable**

Run newcall from the command-line
> .newcall
newcall pauses after the CALL C$DEBUG command.  
At this point, re-enter the Developer Studio.

![Image 2- Newcall is paused](image2)
The Debug Attach function

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

The Reverse Attach Window

From the Reverse Attach Window, select Newcall. Make note of the PID of the program you attach. You may need it later. Click on the Debug button.
The COBOL-IT Debugger Attach

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

Image 4 – The Reverse Attach Windows

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

Click on the “New” button on the toolbar above the panel on the left to create a new configuration.
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.

Image 8- The Main 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.
C/C++ Debug Configuration- Source Tab

Click on the Add button. In the Add Source Windows, select “Path relative to Source Folders”. Click Ok. On the Selection Required screen, select Test3Cobol. Click Ok.

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.

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.
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.
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.
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.

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.

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.

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.