



# isCOBOL™ Evolve

## isCOBOL Evolve 2018 Release 2 Overview

Copyright © 2018 Veryant LLC.

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution and recompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Veryant and its licensors, if any.

Veryant and isCOBOL are trademarks or registered trademarks of Veryant LLC in the U.S. and other countries. All other marks are the property of their respective owners.

## **isCOBOL Evolve 2018 Release 2 Overview**

### **Introduction**

Veryant is pleased to announce the latest release of isCOBOL™ Evolve, isCOBOL Evolve 2018 R2.

isCOBOL Evolve provides a complete environment for development, deployment, maintenance, and modernization of COBOL applications.

isCOBOL 2018 R2 includes several enhancements to GUI controls, such as automatic search and filter in the Grid control, and other new features and compatibility options.

Developer productivity has been enhanced by providing improved code navigation while editing in the IDE and debugging programs, and new debugger features have been implemented.

isCOBOL EIS has been improved to allow tighter integration with external programs.

Support for Asian markets has been improved as well, allowing easy migration to isCOBOL.

Details on these enhancements and updates are included below.

## **isCOBOL IDE Enhancements**

The isCOBOL 2018 R2 IDE improves the editor by displaying useful additional information to COBOL developers and improving the usability of existing features.

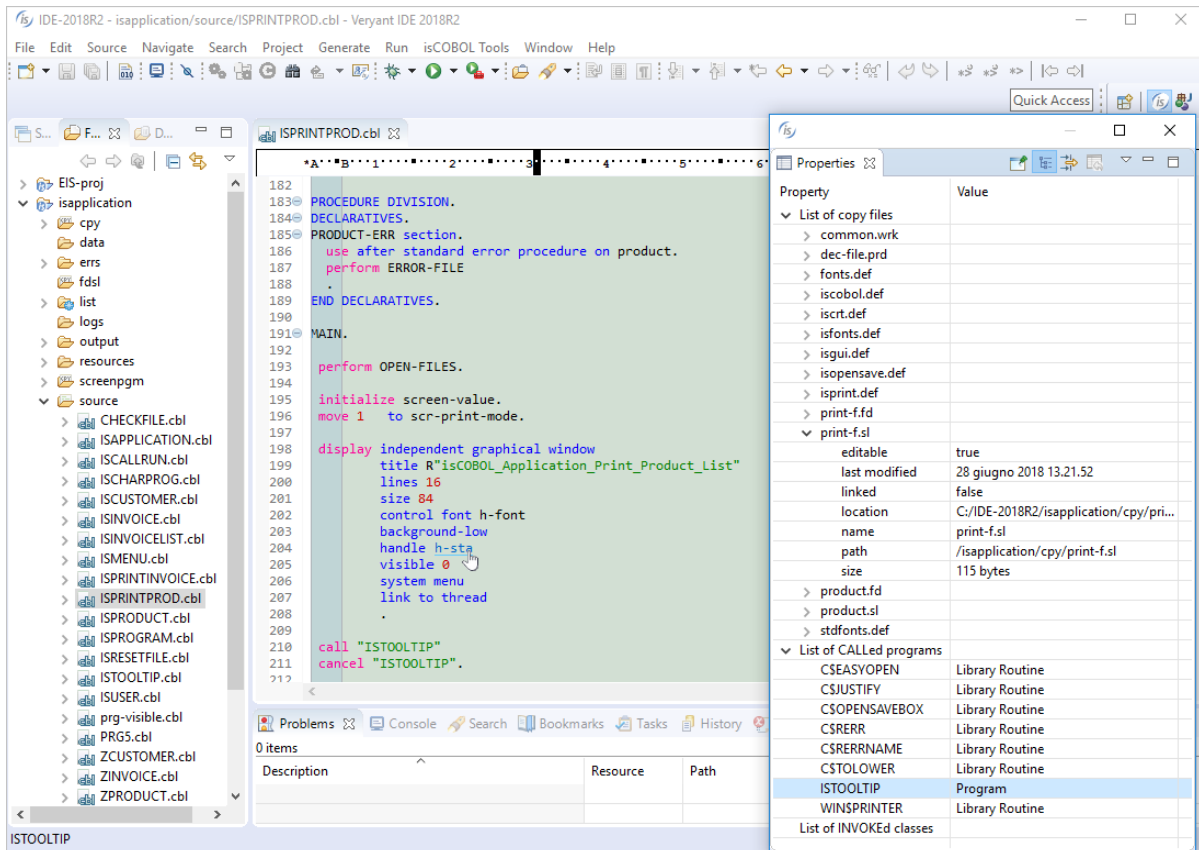
### Editor features

Opening a source file will now list important information in the “Properties” window, by default located on the bottom right of the IDE window, such as:

- a list of copy files used
- a list of called programs
- a list of invoked classes

By double-clicking on a line in the property window, the selected copy file or program will be opened in the Editor if the source is available in the current workspace. For example, double-clicking the selected line shown in Figure 1, *IDE Property window and hyperlink*, will open the “ISTOOLTIP.cbl” source file.

The feature that creates a “Hyperlink” on a variable, paragraph or screen-control name in the editor has been simplified. By clicking on the name when the mouse has the “hand” shaped pointer, the IDE will position the editor on the declaration, whether it is located in the same source file, or in a different copy file.

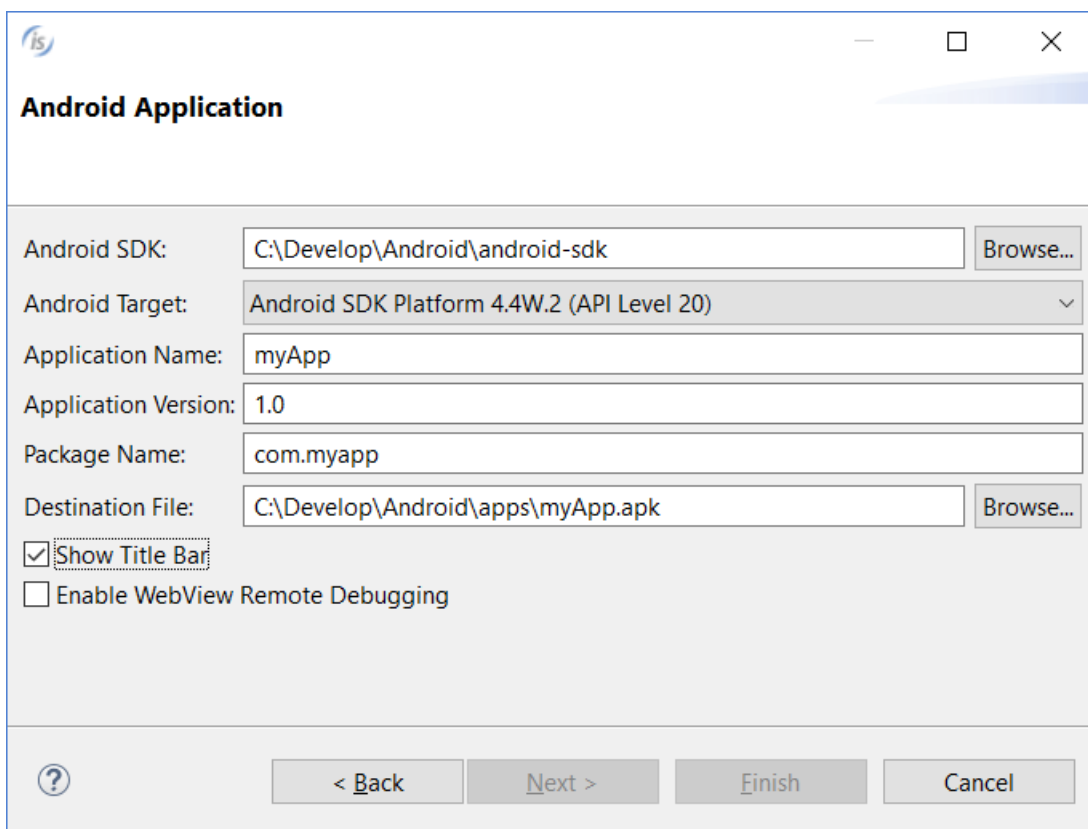
**Figure 1.** IDE Property window and hyperlink

A program can now be compiled even when the current editor file is a copy file, provided the copy file was opened from its parent source file, while previous versions required a program source file to be active.

## Export to Android

The isCOBOL IDE now allows you to customize the application title bar, when exporting the project as an Android application. If you check the “Show Title bar” option as shown in Figure 2, *Android title bar*, then an Application Bar will be rendered at the top of the HTML UI of the exported application, displaying the application name.

**Figure 2.** Android title bar



The screenshot shows a dialog box titled "Android Application" with the following fields and options:

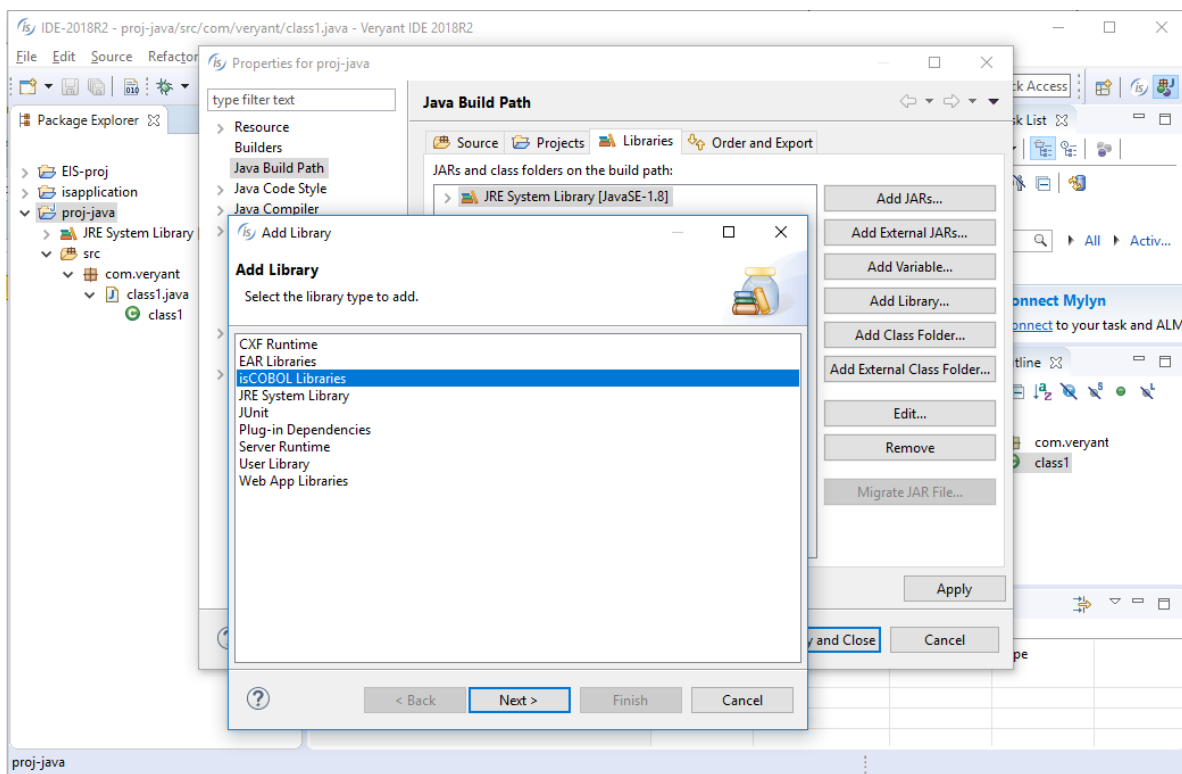
- Android SDK: C:\Develop\Android\android-sdk (with a "Browse..." button)
- Android Target: Android SDK Platform 4.4W.2 (API Level 20) (dropdown menu)
- Application Name: myApp
- Application Version: 1.0
- Package Name: com.myapp
- Destination File: C:\Develop\Android\apps\myApp.apk (with a "Browse..." button)
- Show Title Bar
- Enable WebView Remote Debugging

At the bottom, there are navigation buttons: a help icon (?), "< Back", "Next >", "Finish", and "Cancel".

### Simpler Java integration

The “isCOBOL libraries” item has been added to the “Add Library” button of the “Java Build Path” section in a Java Properties page, allowing simple integration of isCOBOL libraries to a Java application, as shown in Figure 3, *Add isCOBOL Libraries*. This replaces the need to manually add the isCOBOL libraries to the Java project’s ClassPath, required in previous versions.

**Figure 3.** Add isCOBOL Libraries



## **New User Interface Features**

The GRID control has been enhanced with automatic search and filter features. Screen sections are now dynamic, allowing controls to be added and removed at runtime.

Other minor enhancements are designed to improve User Interface handling.

### GRID enhancements

Grid searching is enabled by default and can be activated by the user at runtime by pressing Ctrl-F. A NO-SEARCH style is available to disable the feature when needed.

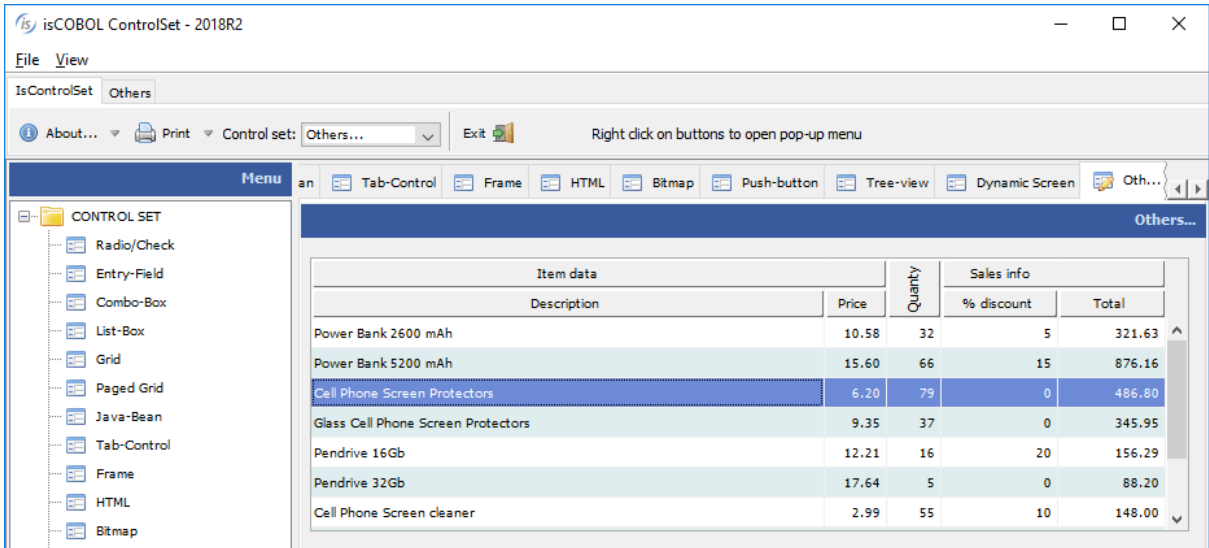
When the search function is activated, a panel is shown on the top portion of the grid to let the user search the grid for data. In the panel the user can enter the search text or select a previously used one. When the Find button is pressed, the occurrences of the search text are highlighted inside the grid. The Clear button allows the user to reset the search text, and the 'X' button allows the user to close the search panel.

The HEADING-MENU-POPUP property of grid now also supports the values GRHM-FIND-ON-RIGHT-CLICK and GRHM-FIND-ON-BUTTON to add the new Find item on the automatic heading menus.

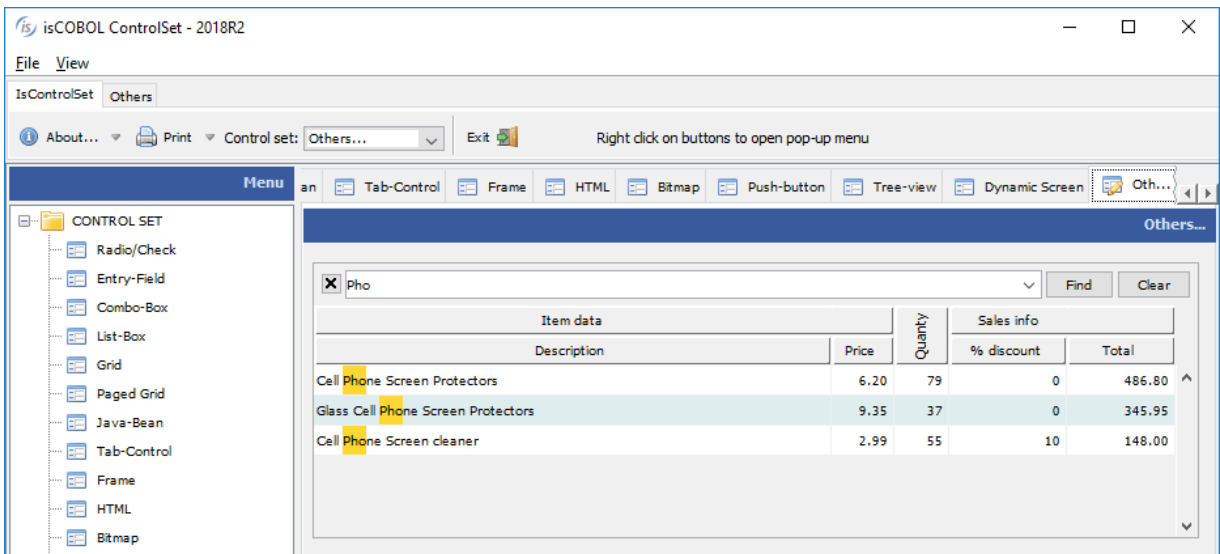
Figure 4, *Grid before activating the search function*, shows the grid while the user is browsing, while Figure 5, *Search on grid* shows the search panel displayed when the user presses Ctrl-F while the grid has focus. This allows the user to search all occurrences of the given text.



**Figure 4.** Grid before activating the search function



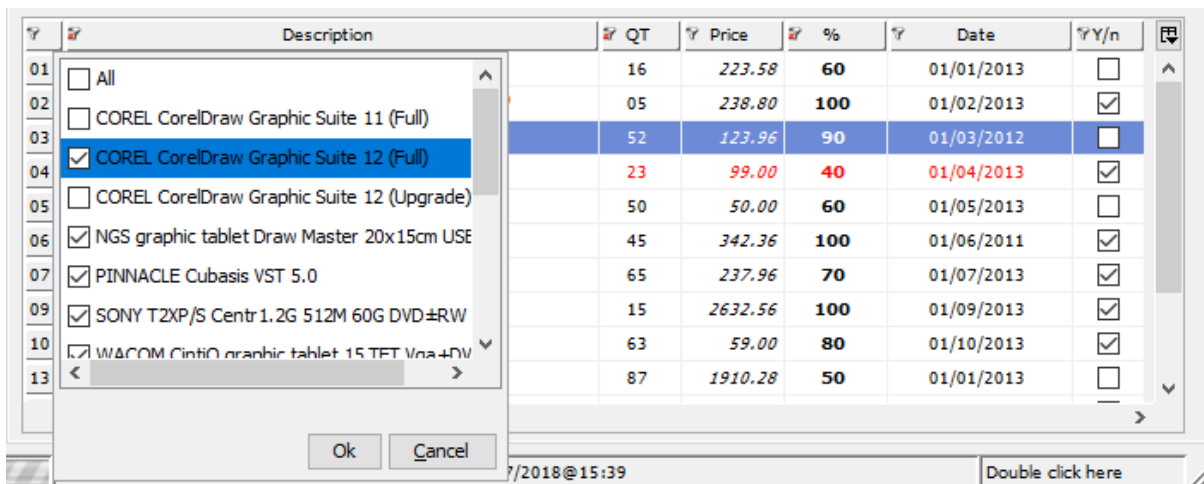
**Figure 5.** Grid with the search function active



A new style named FILTERABLE-COLUMNS is supported on grids, to allow data filtering based on column content, as shown in Figure 6, *Grid filtering*. When activated on a column, a popup window will show every distinct occurrence of the column values, allowing the user to choose one or more values to use for filtering rows. Filters can be added to multiple columns and removed as needed. An icon is displayed on the column header to show an active filter.

```
05 h-grid, grid
   filterable-columns
   ...
```

**Figure 6.** Grid filtering



	Description	QT	Price	%	Date	Y/n
01	All	16	223.58	60	01/01/2013	<input type="checkbox"/>
02	COREL CorelDraw Graphic Suite 11 (Full)	05	238.80	100	01/02/2013	<input checked="" type="checkbox"/>
03	<input checked="" type="checkbox"/> COREL CorelDraw Graphic Suite 12 (Full)	52	123.96	90	01/03/2012	<input type="checkbox"/>
04	COREL CorelDraw Graphic Suite 12 (Upgrade)	23	99.00	40	01/04/2013	<input checked="" type="checkbox"/>
05	COREL CorelDraw Graphic Suite 12 (Upgrade)	50	50.00	60	01/05/2013	<input type="checkbox"/>
06	<input checked="" type="checkbox"/> NGS graphic tablet Draw Master 20x15cm USE	45	342.36	100	01/06/2011	<input checked="" type="checkbox"/>
07	<input checked="" type="checkbox"/> PINNACLE Cubasis VST 5.0	65	237.96	70	01/07/2013	<input checked="" type="checkbox"/>
09	<input checked="" type="checkbox"/> SONY TZXP/S Centr 1.2G 512M 60G DVD±RW	15	2632.56	100	01/09/2013	<input checked="" type="checkbox"/>
10	<input checked="" type="checkbox"/> WACOM Cintiq graphic tablet 15 TFT VGA+FW	63	59.00	80	01/10/2013	<input checked="" type="checkbox"/>
13	< >	87	1910.28	50	01/01/2013	<input type="checkbox"/>

## Dynamic screens

Screen sections are now dynamic, allowing controls or entire screen to be added or removed at runtime using the DISPLAY UPON SCREEN and DESTROY syntax.

The UPON target can be a screen section, using the following syntax

```
display screen1 upon screen2
```

or can be any of its group levels, allowing creation of child controls, using the syntax

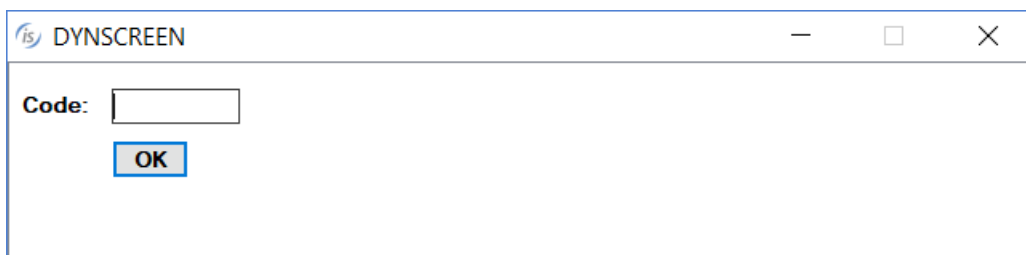
```
display screen1 upon screen2-group-2
```

The following code displays a screen with only a push-button, and populates it at runtime using DISPLAY statements, before accepting user input.

```
program-id. dynscreen.
working-storage section.
77 key-status special-names crt status pic 9(4).
88 exit-pushed value 27.
77 w-code pic x any length.
screen section.
01 screen1.
05 push-button ok-button
line 4 col 9, size 6 cells.
procedure division.
main.
*display a window
display standard graphical window.
*display the initial screen
display screen1.
*add a label
display label upon screen1
line 2, col 2, size 6 cells
title "Code:".
*add an entry-field
display entry-field upon screen1
line 2, col 9, size 10 cells
value w-code.
*accept the screen
perform until exit-pushed
accept screen1
on exception
continue
end-accept
end-perform.
destroy screen1.
```

The result of running the code above is shown on Figure 7, *Dynamic screen*

**Figure 7.** Dynamic screen



#### Other enhancements

The message-box control now plays a corresponding notification sound based on the message type, more closely adhering to the Microsoft® Windows® standard.

## **Framework improvements**

The new release features a new modeless print preview window. Prior to version 2018R2 print preview windows were modal.

The new audit feature allows the auditing of the I/O processing taking place in the application, without requiring code changes.

### Print Preview

isCOBOL Print Preview is now a modeless window allowing multiple simultaneous preview windows. The user can switch between opened preview windows by clicking on them.

### Audit feature

The audit feature builds on top of the I/O Trigger file technology, allowing COBOL developers to add logging on all I/O operations without source code modifications.

The isCOBOL 2018R2 GUI tool provides an easy way to configure the auditing features - specifying users, operations, and files on which auditing is triggered. Auditing can be configured to trigger on all users accessing one or more files, or on specific operations such as delete operations executed from users on a specific file.

The auditing feature is written in COBOL and provided as source code, to be easily customized to any environment and use case.

## Compatibility improvements

Additional ESQL syntax supported in Pro\*COBOL is now supported in isCOBOL, simplifying migration from Pro\*COBOL to JDBC.

New syntax, library routines and configuration properties are supported to enhance compatibility with other COBOL dialects.

### New ESQL syntax:

- Support for ESQL LOCK TABLE statement to acquire a lock on a table or portion of a table.

Code snippet:

```
exec sql
    lock table customers
        in row exclusive mode
        nowait
end-exec.
```

- The FOR clause and the use of array items (OCCURS) without an index is now supported on several ESQL statements. The runtime repeats a statement multiple times when an OCCURS data item is used among host variables. For example, the following code:

```
working-storage section.
77 ins-values pic x(10) occurs 4.
procedure division.
    move "aaa" to ins-values(1).
    move "bbb" to ins-values(2).
    move "ccc" to ins-values(3).
    move "ddd" to ins-values(4).
    exec sql
        for 3
            insert into tbl1 (column1) values (:ins-values)
    end-exec.
```

will cause the runtime to repeat the INSERT statement 3 times, using ins-values(1), ins-values(2) and ins-values(3) respectively. Without the FOR clause, the statement would be executed 4 times, one for each item in ins-values.

- The RETURNING clause is now supported, allowing retrieval of updated values after a statement has been executed, for example:

```
exec sql
  insert into emp1
    (empno,
     ename,
     deptno
    )
  values
    ('12', 'John Doe', 'dep1')
  returning empno, ename, deptno into
    :new_emp_number, :new_emp_name, :new_dept
end-exec.
```

- The INTO clause of SELECT and FETCH ESQL statements can now be bound to an OCCURS data item, for example:

```
working-storage section.
77 col-val pic x(10) occurs 3.
procedure division.
  exec sql
    select column1 into :col-val from tbl1
  end-exec.
```

The above statement will read 5 records from table tbl1 and store the values of field column1 in col-val(1), col-val(2), col-val(3).

### Improved COBOL compatibility

isCOBOL now provides even better compatibility with other COBOL dialects.

- Properties in CLASS-ID programs are now supported. The CLASS-ID program sets the property as it would do with a standard variable, for example:

```
identification division.  
Class-id. myClass.  
identification division.  
factory.  
working-storage section.  
01 myProp pic 9(10) comp property.  
procedure division.  
identification division.  
method-id. myMethod  
procedure division.  
main.  
    move 2 to myProp.
```

Legacy programs and CLASS-ID programs that reference the class above can also reference the property in the REPOSITORY paragraph, and use it as a standard COBOL variable, for example:

```
configuration section.  
repository.  
    class myClass as "myClass"  
    property myProp.  
procedure division.  
main.  
    display myProp of myClass.
```

- ACTIVE-CLASS syntax is now supported, allowing you to identify a specific instance of the current class or one of its subclasses.

Code snippet:

```
method-id. getInstance as "getInstance".  
working-storage section.  
77 cls-instance object reference active-class.  
procedure division returning cls-instance.  
main.  
    invoke self "new" giving cls-instance.
```



- New Micro Focus® directives are now supported:

```
$SET INDD"<filename>"  
$SET OUTDD"<filename [recsize] [filetype]>"
```

To enable you to map the system input (SYSIN) and system output (SYSOUT) to disk files. Each program can use different disk files. This is explained in detail later in this document.

### New library routine

A new library routine has been implemented to provide better compatibility with RM/COBOL®:

**C\$WRU** returns the name of the calling program.

The following code snippet shows the usage of the C\$WRU library routine

```
01 WHO-CALLED-ME .  
    05 THE-CALLING-PROGRAM PIC X(30) VALUE SPACES .  
    05 THE-CALLING-LINE    PIC S9(6)  BINARY .  
    05 THE-LINE-NUM       PIC S9(02) BINARY .  
PROCEDURE DIVISION .  
    CALL 'C$WRU' USING THE-CALLING-PROGRAM  
                      THE-CALLING-LINE  
                      THE-LINE-NUM .
```

### New configuration property

The new `iscobol.memory.alpha_edited=true` configuration property has been implemented to manage the VALUE clause of alphanumeric edited items in compatibility with Micro Focus®, AcuCOBOL-GT® and RM/COBOL®.

## isCOBOL Compiler Enhancements

isCOBOL Evolve 2018 R2 implements new compiler options and new syntax to better support COBOL applications that use DBCS (Double Byte Character Strings) such as Japanese, Chinese and Korean without Unicode encoding.

The Compiler can now automatically compile classes used by the program being compiled.

### New compiler options

-ccbas to count bytes instead of characters for fixed (aka ANSI) source code

This option helps when compiling source files that contain DBCS text. By default, the isCOBOL compiler counts characters in order to determine the text positions of the various areas of the Fixed (ANSI) COBOL source format, while Asian compilers count the bytes instead. Source files written for Asian COBOLs may not cleanly compile without the -ccbas option.

-cndbc to use the DBCS instead of Unicode in PIC N without the USAGE NATIONAL clause.

A data item such as:

```
77 n-item pic n(10).
```

stores data in UTF-16 Big Endian by default. When compiling the program with -cndbc data will be stored using the current encoding. If both Unicode and DBCS data items are needed in the same application, they can be declared as follows:

```
77 dbcs-item pic n(10).  
77 unicode-item pic n(10) USAGE NATIONAL.
```

### Automatic class compilation

When you compile COBOL programs (PROGRAM-ID) or COBOL classes (CLASS-ID) that reference additional COBOL classes (CLASS-ID), the Compiler now automatically compiles the referenced source code.

For example, considering the following three source code files:

Prog.cbl:

```
identification division.  
program-id. Prog.  
environment division.  
configuration section.  
repository.  
    class Class2 as "Class2".  
procedure division.  
main.  
    Class2:>method1().  
    goback.
```

Class1.cbl:

```
identification division.  
class-id. Class1 as "Class1".  
identification division.  
factory.  
procedure division.  
identification division.  
method-id. method1 as "method1".  
procedure division.  
main.  
* do something here  
end method.  
end factory.
```

Class2.cbl:

```
identification division.  
class-id. Class2 as "Class2" inherits Class1.  
environment division.  
configuration section.  
repository.  
    class Class1 as "Class1".  
identification division.  
factory.  
procedure division.  
*add some methods here  
end factory.
```

When compiling Prog.cbl, the Class1.cbl and Class2.cbl source files are automatically compiled, both when compiling from the IDE or from the command line.

### SYSIN and SYSOUT mapping

Two new compiler properties have been added to allow you to map the system's standard input (SYSIN) and standard output (SYSOUT) to disk files.

```
iscobol.compiler.indd=<filename>
iscobol.compiler.outdd=<filename [recsize] [filetype]>
```

For example, let's consider using the following compiler configuration properties:

```
iscobol.compiler.indd=input.txt
iscobol.compiler.outdd=output.txt
```

If a program is compiled with those properties, it will read a line from the file `input.txt` instead of accepting the user input when it performs an *ACCEPT dest-item FROM SYSIN*. When the program performs a *DISPLAY ... UPON SYSOUT*, it will write a line to the file `output.txt` instead of printing on the console.

To allow programs to use program-specific input and output files, these two properties can also be used as compiler directives inside the source code. For example:

```
$SET INDD "input-prog1.txt"
$SET OUTDD "output-prog1.txt"
program-id. prog1.
```

## isCOBOL Debugger

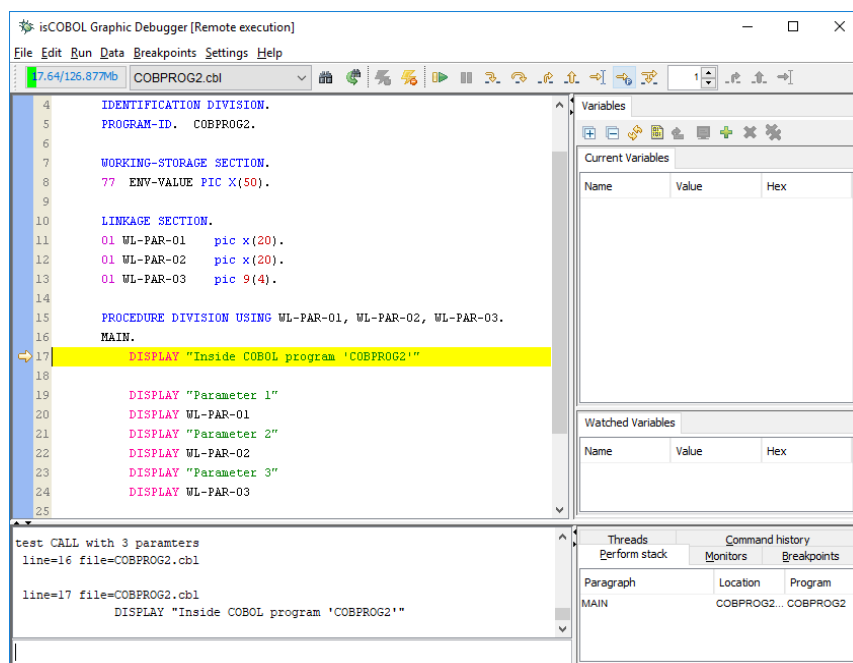
The isCOBOL Debugger has greatly enhanced, with new features aimed at increasing developer productivity, such as a new debugger command, easier code navigation and more.

### New debugger command

The isCOBOL Debugger has been enhanced with the new feature “*Run to next program*”, using the command PROG. When the PROG command is used, the Debugger continues execution of the current program, stopping at the beginning of the next COBOL program. This is especially useful when the caller program is not a COBOL program, such as a Java or C program, allowing the debugger to stop at the first COBOL program called.

For example, by typing PROG in the command window, pressing the button on the toolbar shown in Figure 8, or choosing the option from the menu, *Debugger command PROG*, the COBPROG2 will be executed and the Debugger will stop when running COBPROG3, which is the next program called by the main Java source.

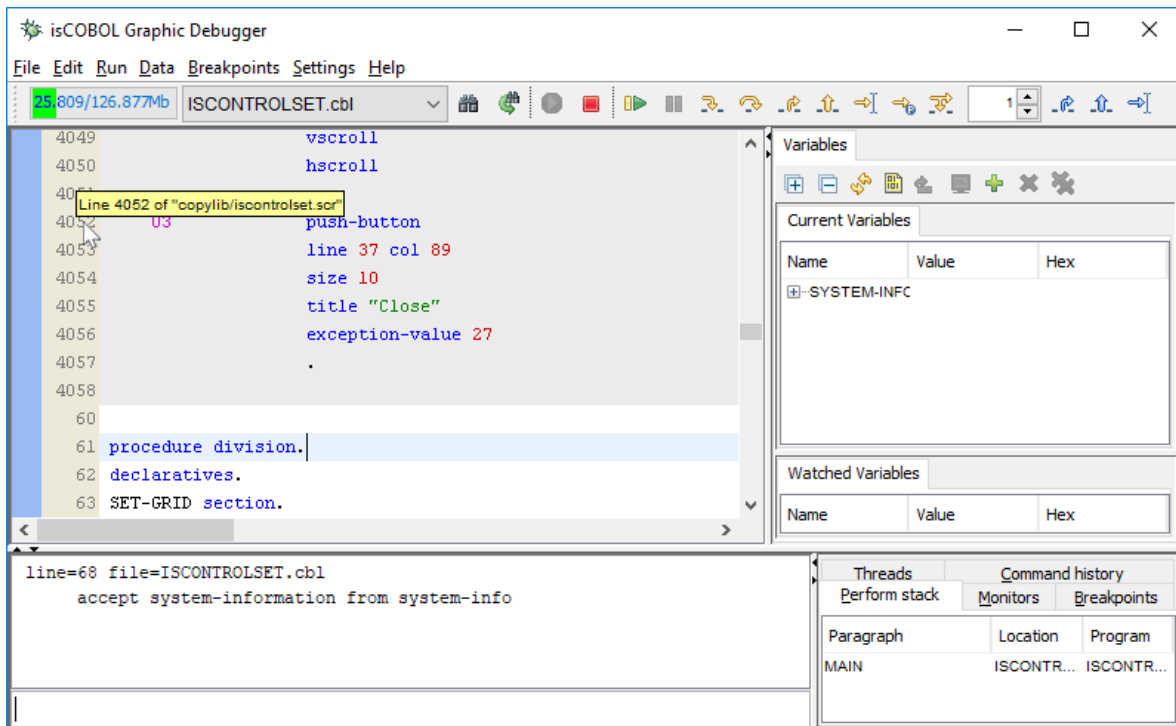
**Figure 8.** Debugger command PROG



### Hint on line column

In large programs with many copybook files, sometimes it's difficult to determine which file contains the source code being debugged. With the new debugger, a hint will appear when hovering the mouse pointer over the line number column, located on the left pane of the Debugger window, displaying the file path and line number of the highlighted line. Figure 9, *Debugger hint on line column*, shows the feature in action.

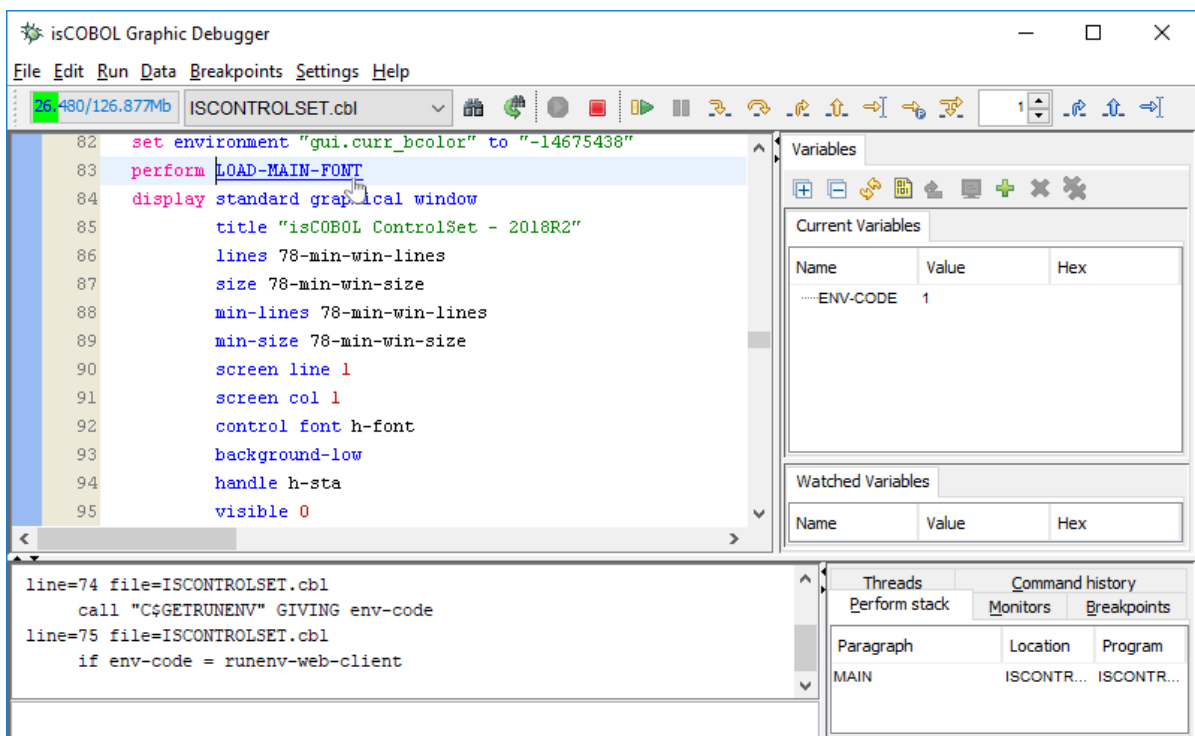
**Figure 9.** Debugger hint on line column



## Hyperlink

To simplify jumping to a given paragraph or variable definition, the hyperlink feature has been introduced. When hovering the mouse pointer on a paragraph name or a variable name, the name will be underlined and the mouse cursor will change to hand shape point, as shown in Figure 10, *Debugger hyperlink*. Left clicking the name will cause the debugger to display the definition.

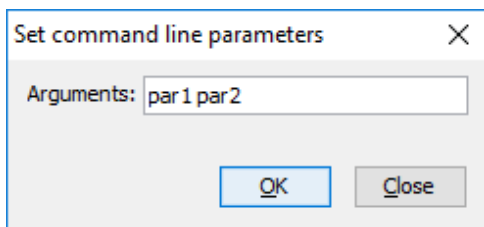
**Figure 10.** Debugger hyperlink



### Command line parameters

The 'Set command line parameters' values are now saved in the Debugger session file (.isd), making it easier to debug a program with CHAINING parameters in different debugging sessions, as shown in Figure 11, *Debugger Set command line parameters*.

**Figure 11.** Debugger Set command line parameters





## isCOBOL EIS

The isCOBOL EIS features have been improved in several areas, such as the HTTPHandler class, WD2 routines, and Stream2Wrk utility.

### HTTPHandler improvements

isCOBOL EIS now allows access to key Servlet objects in both COBOL Servlets and Web Direct 2.0 applications.

- The HTTPHandler class provides the following new methods:

```
HTTPHandler:>getRequest()
```

```
HTTPHandler:>getResponse()
```

```
HTTPHandler:>getSession()
```

These methods return the instance of the HTTP Request, Response and Session respectively. For example, these objects can be used to retrieve the IP address of the end user in your Servlet, using the following code:

```
repository.  
    class HTTPHandler as "com.iscobol.rts.HTTPHandler"  
    class HTTPRequest as "javax.servlet.ServletRequest"  
.  
working-storage section.  
77 servlet-request object reference HTTPRequest.  
77 client-ip pic x any length.  
linkage section.  
77 http-handler object reference HTTPHandler.  
procedure division using http-handler.  
    set servlet-request to http-handler:>getRequest()  
        as HTTPRequest.  
    set client-ip to servlet-request:>getRemoteAddr().
```

### WD2 improvements

The Web Direct 2.0 routine WD2\$SESSION has been enhanced with new properties that return information about the servlet context and the HTTP session of the servlet managing the COBOL application. They are:

```
iscobol.wd2.servletcontext.name  
iscobol.wd2.servletcontext.realpath  
iscobol.wd2.servletcontext.path  
iscobol.wd2.servletcontext.serverinfo  
iscobol.wd2.servletcontext.majorversion  
iscobol.wd2.servletcontext.minorversion  
iscobol.wd2.httpsession.id  
iscobol.wd2.httpsession.creationtime
```

For example, to retrieve the real path of an application deployed in a servlet container, the following code can be used:

```
working-storage section.  
copy "iscobol.def".  
77 real-path pic x any length.  
procedure division.  
    call "wd2$session" using wd2-get-session-value  
                            "iscobol.wd2.servletcontext.realpath"  
                            real-path.
```

Also, the rendering of controls in WD2 has been upgraded to improve screen section display.

### Stream2Wrk improvements

The Stream2Wrk utility provides a new option to specify the name of the root element when processing JSON streams that have no root element.

For example, the following JSON:

```
{
  "name": "John",
  "age": 30,
  "cars": [ "Ford", "BMW", "Fiat" ]
}
```

would generate the following working-storage structure:

```
01 json2wrk identified by ''.
   03 name identified by 'name'.
       05 name-data pic x any length.
   03 age identified by 'age'.
       05 age-data pic x any length.
   03 cars identified by 'cars' occurs dynamic capacity cars-count.
       05 cars-data pic x any length.
```

To change, for example, the automatically generated “json2wrk” name created by Stream2Wrk to “allcars”, the following command line can be used:

```
$ stream2wrk json yourfile.json -r allcars
```

The result will be:

```
01 allcars identified by ''.
   03 name identified by 'name'.
       05 name-data pic x any length.
   03 age identified by 'age'.
       05 age-data pic x any length.
   03 cars identified by 'cars' occurs dynamic capacity cars-count.
       05 cars-data pic x any length.
```