isCOBOL™ Evolve

isCOBOL Evolve 2020 Release 2 Overview

isCOBOL Evolve 2020 Release 2 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL™ Evolve, isCOBOL Evolve 2020 R2.

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

A new isCOBOL Profiler makes it easier to identify application bottlenecks.

The 2020 R2 release has new GUI controls such as the new “hamburger menu” and glyph fonts rendering, and improves many existing ones, such as grids and message boxes, to further help developers modernize their applications.

The Debugger has been reengineered, and now source code can be embedded in the compiled class, making debugging sessions possible without having the source. The new debugger also adds a new Console View that separates messages coming from sysout and syserr from other debugger output.

Details on these enhancements and updates are included below.

New isCOBOL Profiler

Starting with the 2020R2 Release, a new Profiler has been implemented to allow performance analysis on applications, making it easy to identify which program or paragraphs take longer to execute. Developers can view the gathered information in a clean HTML output, and take actions to perform code optimizations accordingly to maximize gains.

This feature is now available in the isCOBOL Evolve suite, and can be accessed from the command line or from the Eclipse-based isCOBOL IDE.

The Profiler can be enabled in different ways depending on deployment architecture. It can be activated using a runtime option, as shown in the command line below:

```
iscrun –profile IO_PERFORMANCE
```

With this command, the isCOBOL Profiler creates a folder called "hprofHtmlReport" containing an HTML report of the profile analysis. Figure 1, *isCOBOL Profile report,* shows the report with a list of all executed programs and the percentage of the total run time, as well as individual paragraph profile results.

The report shows both the percentage of total time and the number of seconds spent in each programs' paragraphs. It also shows how many times a paragraph has been executed in the "Count" column.

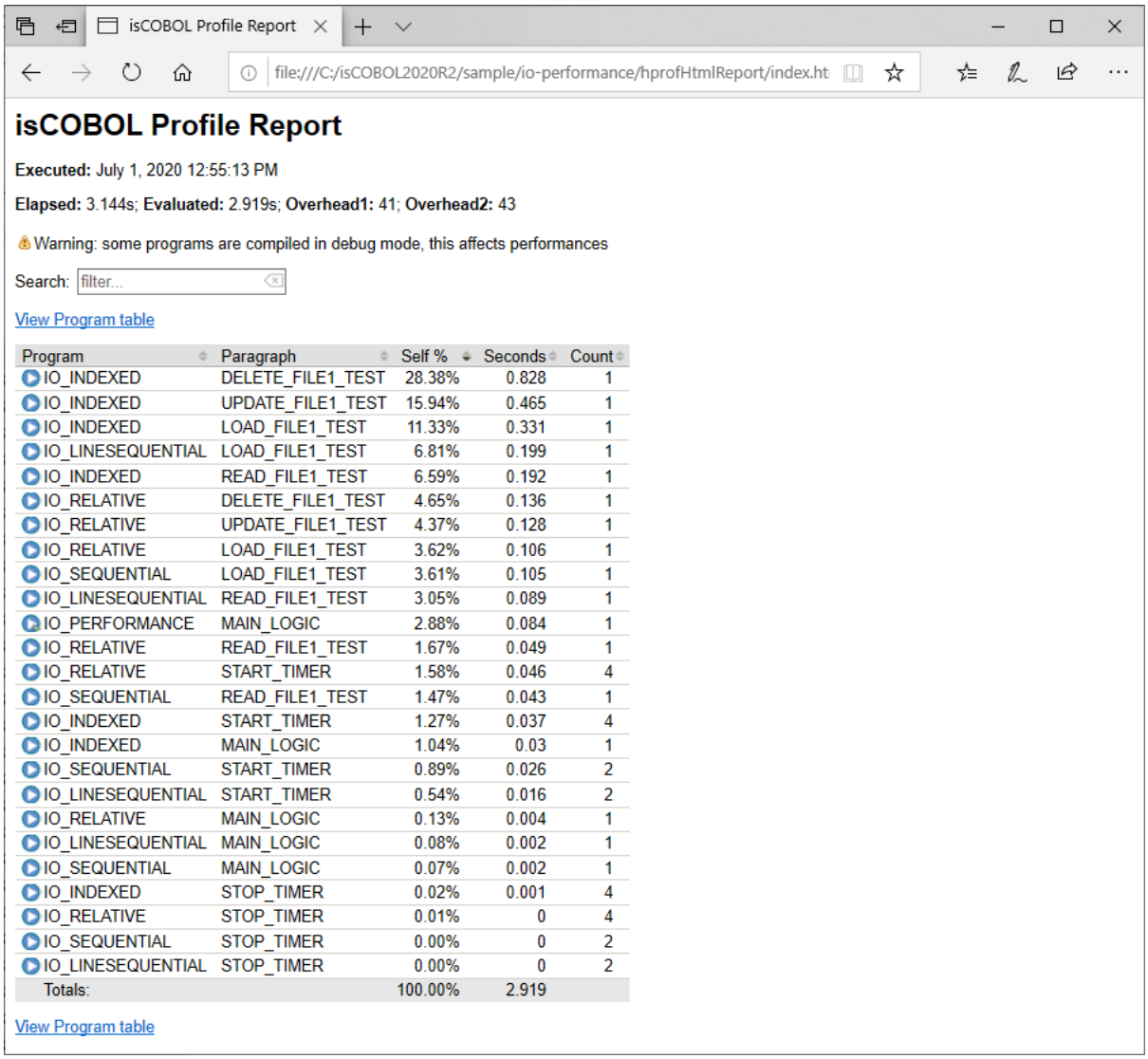
Another way to start the profiling process, compatible with the previous profiler feature, is using a java option:

```
-javaagent:/Veryant/isCOBOL2020R2/lib/isprofiler.jar
```

This method of activating the profiler is most useful when running in an Application Server architecture, or in scenarios such as Java programs calling isCOBOL or running isCOBOL EIS applications in a J2EE container like Apache Tomcat, when the main class is not the COBOL program.

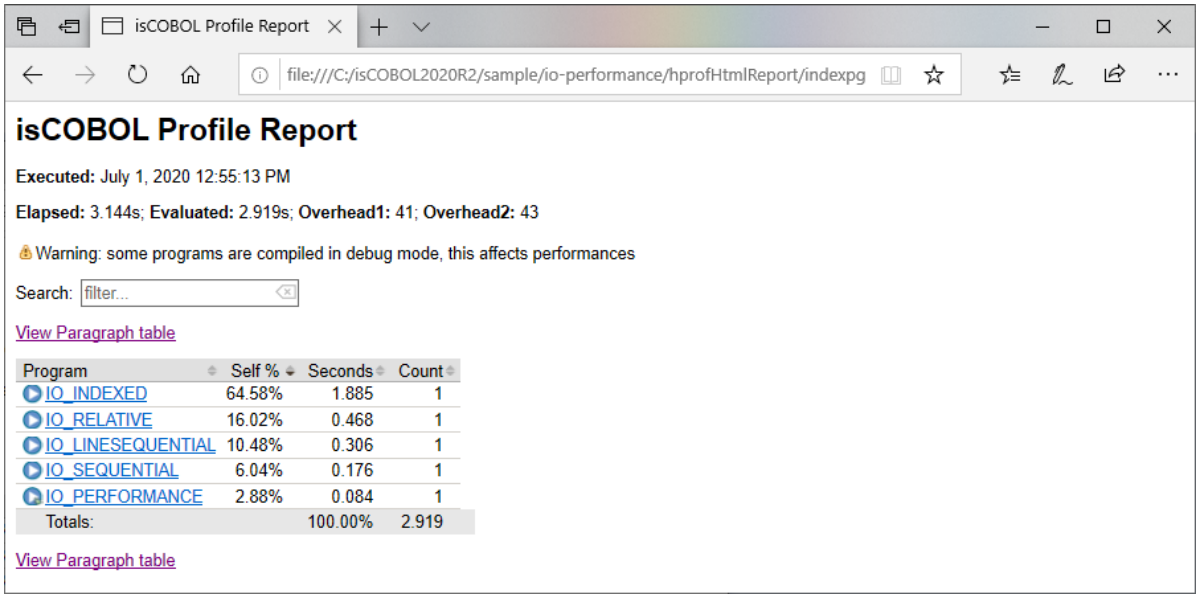
In isCOBOL 2020 R2, the Profiler can also be activated or stopped dynamically at runtime, by performing a CALL "C$PROFILE" instruction. An example of this usage will be shown in a later section of this document.

Figure 1. isCOBOL Profile report



When clicking on the "View Program table" link shown in the figure above, the profiler report is opened, showing the profiling details of each program without the paragraph details, as shown in Figure 2, *isCOBOL Profile program table*.

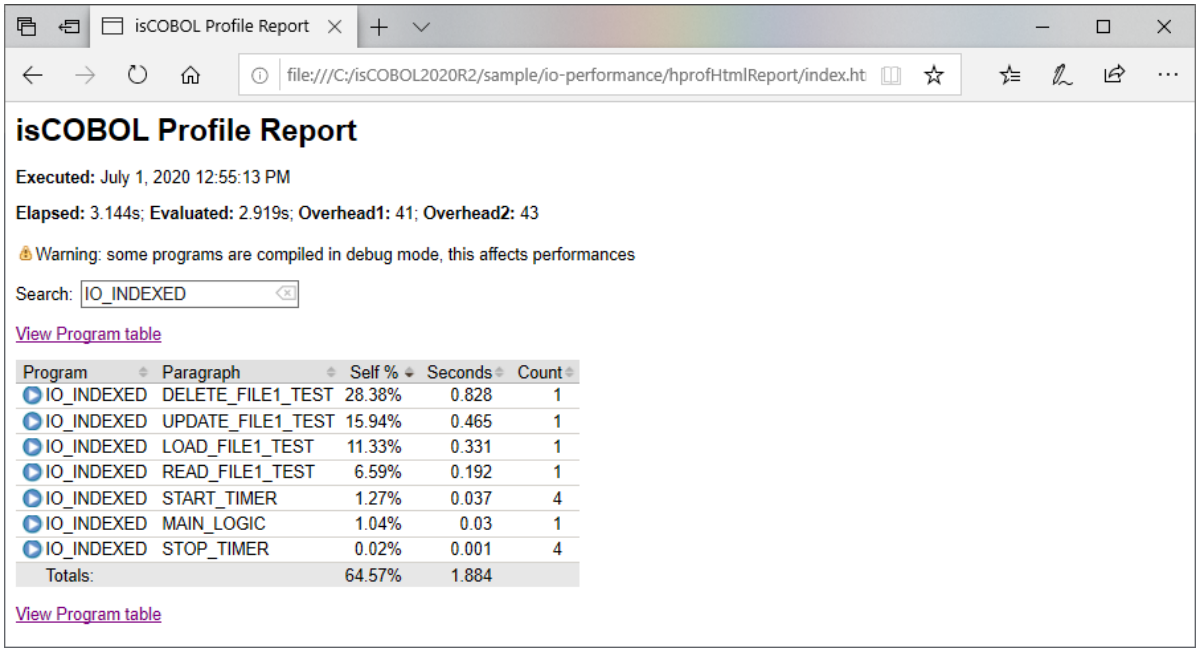
Figure 2. isCOBOL Profile program table



Clicking on a source file name, for example IO_INDEXED, the Paragraph table view is loaded, with the filter automatically set to the selected program, thus showing all its paragraphs performance details.

In Figure 3, *isCOBOL Profile filter,* shows how to filter the output by program name.

Figure 3. isCOBOL Profile filter



The report can be filtered by typing in the Search field, to select specific programs or paragraphs. The filter is applied to all columns in the table. The report shows a warning if programs are profiled when compiled in debug mode, or if logging is enabled, as that will impact performance. Should this occur, developers should recompile the programs without the debug option, and run with the iscobol.tracelevel configuration option disabled in order to profile with maximum performance.

Profiler reports can be customized using new configuration options:

- `iscobol.profiler.html=path` sets the folder in which isCOBOL Profiler will create the report. If not set, the default value is "./hprofHtmlReport"
- `iscobol.profiler.xml=path` sets the file path of xml report. If not set, the xml report is not generated. This is needed when running the isCOBOL Profiler inside the IDE to take advantage of its specific View.
- `iscobol.profiler.excludes=path` sets the list of isCOBOL classes that will be excluded from the profiling analysis. It is a comma-separated list of regular expressions defining the programs, for example IO_INDEXED,IO_RELATIV[A-Z_]
- `iscobol.profiler.includes=path` sets the list of isCOBOL classes that will be included in the profiling analysis. It uses the same syntax explained above.

The new library routine C$PROFILE is handy when trying to profile a program that contains ACCEPT statements, since performance analysis is affected by the time the user takes to complete the ACCEPT statement. Developers can easily suspend the profiling process when such events occur, and resume it when it's done. The routine also allows you to dynamically set the report format and the folder in which to create the report.

The code snippet below shows how to perform all the steps explained above from code:

```
call "C$PROFILER" USING cprof-disable
accept mask-parameters
call "C$PROFILER" using cprof-set "html" "html-profiler-output"
call "C$PROFILER" USING cprof-enable
perform elaboration.
call "C$PROFILER" USING cprof-disable
display message "processing completed"
call "C$PROFILER" USING cprof-flush
```

The Code Coverage options have analogous settings to control its behavior at runtime using the C$COVERAGE library routine with new op-codes, as shown in the snippet below:

```
call "C$COVERAGE" using ccov-set "html" "html-coverage-output"
...
call "C$COVERAGE" USING ccov-flush
```

The Code Coverage feature, which was previously available when running stand-alone programs using iscrun, is now fully supported in other environments, such as the Application Server, using the new property options for –javaagent, as shown below:

```
-javaagent:/Veryant/isCOBOL2020R2/lib/isprofiler.jar=coverage;html=html-coverage;profiler;html=html-profiler
```

When run with this option, the Application Server, upon termination, will create the Code Coverage and the Profiler reports in two different folders, unless the C$PROFILER or C$COVERAGE routines are called by running programs to flush the reports.

The –javaagent option enables the customization of additional parameters of both Code Coverage and Profiler features:

- **includes=** comma separated list of regular expressions to specify the classes to be included in the report
- **excludes=** comma separated list of regular expressions to specify the classes to be excluded from the report
- **xml=** name of the xml output file

Additional options are available for the Coverage feature:

- **sourcefiles=** the location of the source files
- **classfiles=** the location of the class files
- **append=** the xml file to be appended to the output. This can be specified multiple times
- **sessionname=** the name of the coverage session

All these features enhance Veryant products and complete the isCOBOL Code Coverage and isCOBOL Unit Test features introduced in the 2020 R1 release, and are especially appealing to enterprise users.

isCOBOL IDE enhancements

The Profiler feature can now be executed from the isCOBOL 2020 R2 IDE, and a specific View is provided to show and analyze the results of the profiling process.

Figure 6, *isCOBOL Profiler in isCOBOL IDE* shows how to enable profiling on specific programs. By default, all programs are included in Profiler, and the results gathered by the analysis are available in the Profiler View, as shown in Figure 7, *isCOBOL Profiler View*.

Figure 6. isCOBOL Profiler in isCOBOL IDE

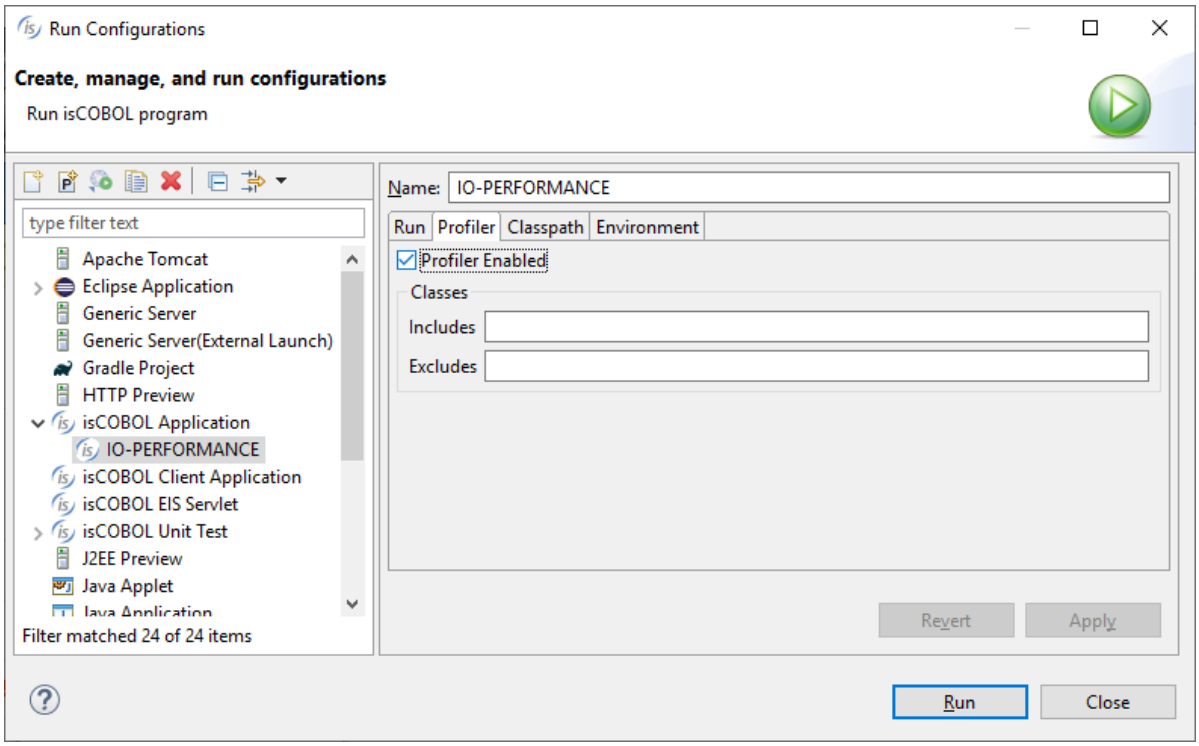
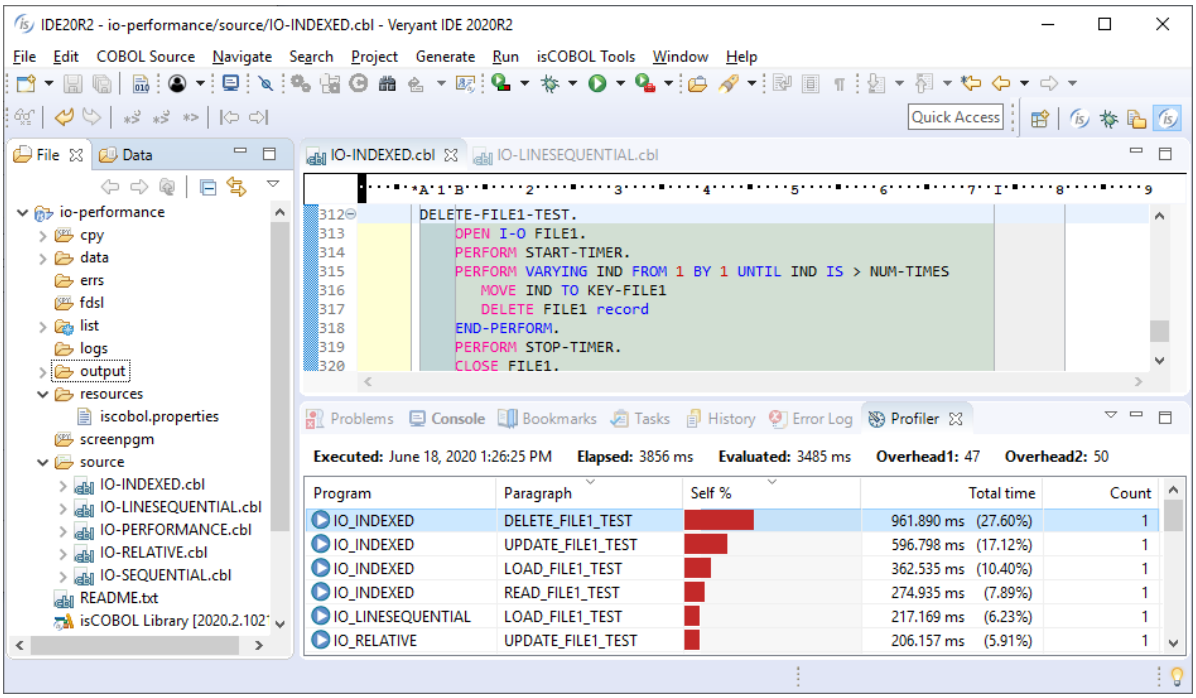


Figure 7. isCOBOL Profiler View



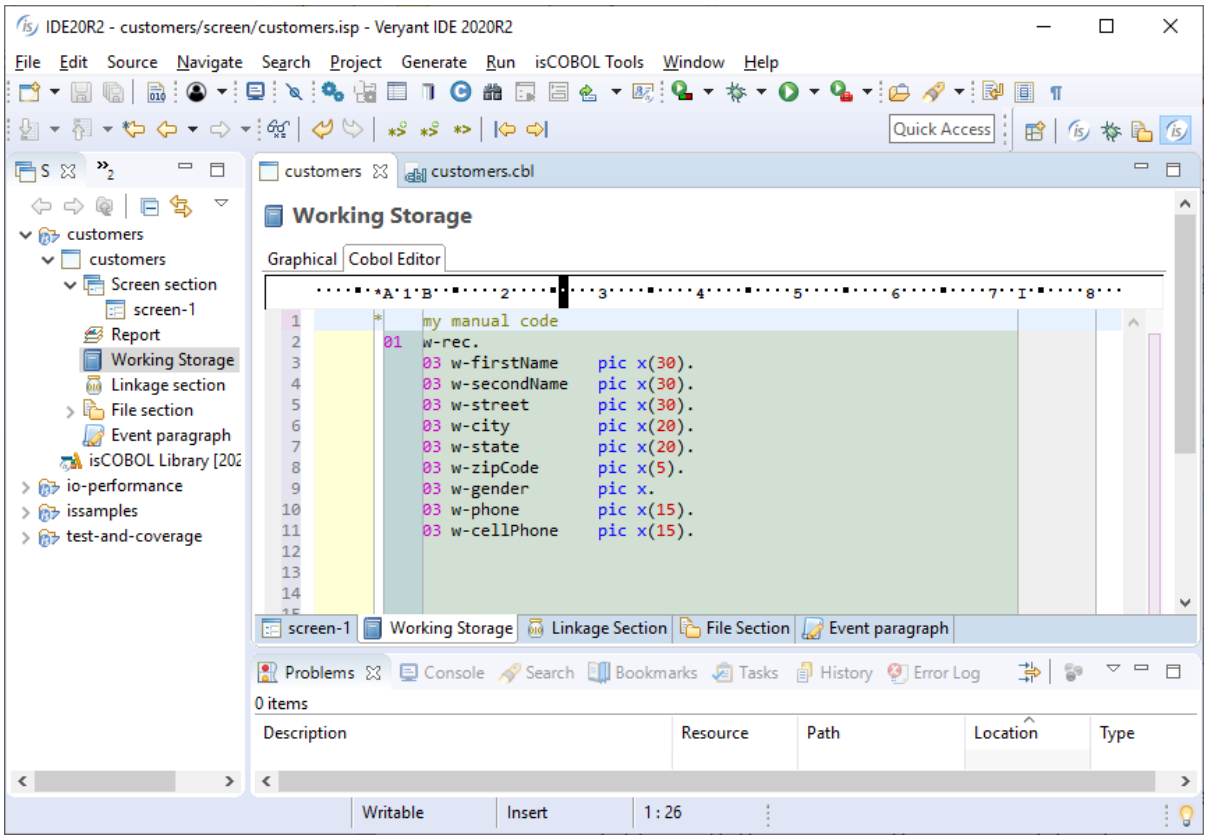
Double clicking an item in the Profiler View opens the selected program in the Editor, with the cursor located at the specific paragraph, helping developers speed up the process of identifying bottlenecks to optimize the code for better performance.

The isCOBOL IDE's Screen Painter now supports manual editing in Working Storage, Linkage Section and FD sections, to complement the Graphical editor previously available.

Changes made manually are available in the Screen Painter immediately. This feature will be appreciated by COBOL developers who would prefer to code by hand instead of through the Graphical editor.

Figure 8, *isCOBOL IDE Cobol Editor*, shows the Cobol Editor section of the Working Storage painter, where a group level variable is manually coded. The Painter will append the manual code fragment in the generated source code after the variables defined in the Graphical editor.

Figure 8. isCOBOL IDE Cobol Editor



GUI enhancements

GUI controls have been improved in the 2020 R2 release. A new menu style is now available, the Hamburger menu, that arranges its items in a tree-view style.

The W$BITMAP library routine can now render glyph font characters as images that can be used as icons in any control that can handle bitmaps.

Hamburger menu

The isCOBOL compiler now supports a new menu style that arranges items in a hierarchical tree-view which appears after pressing an "hamburger" icon, and can replace the classic menu bar. This new feature can be used to give applications a more modern look, or to replace a standard menu bar in a responsive application to adapt to a resized surface area.

A new op-code, `wmenu-new-hamburger`, has been added to the W$MENU library routine to enable the creation of hamburger menus.

A code snippet to create a hamburger menu is shown below:

```
call "W$MENU" using wmenu-set-attribute
                    "default-font" h-menu-font
call "W$MENU" using wmenu-set-attribute
                    "width" 160
call "W$MENU" using wmenu-new-hamburger giving menu-handle
```

The new `wmenu-set-attribute` op-code allows full customization of the graphical appearance of the menu, such as fonts, colors and icons. The full list of supported attributes is: `check-icon, default-background-color, default-font, default-text-color, disabled-background-color, disabled-font, disabled-text-color, dropdown-icon, dropdown-open-icon, hamburger-icon, hamburger-open-icon, hover-background-color, hover-font, hover-text-color, position, tool-bar-covering, style, width`.

Existing W$MENU op-codes can be used to manage the hamburger menu the same way as regular menus.

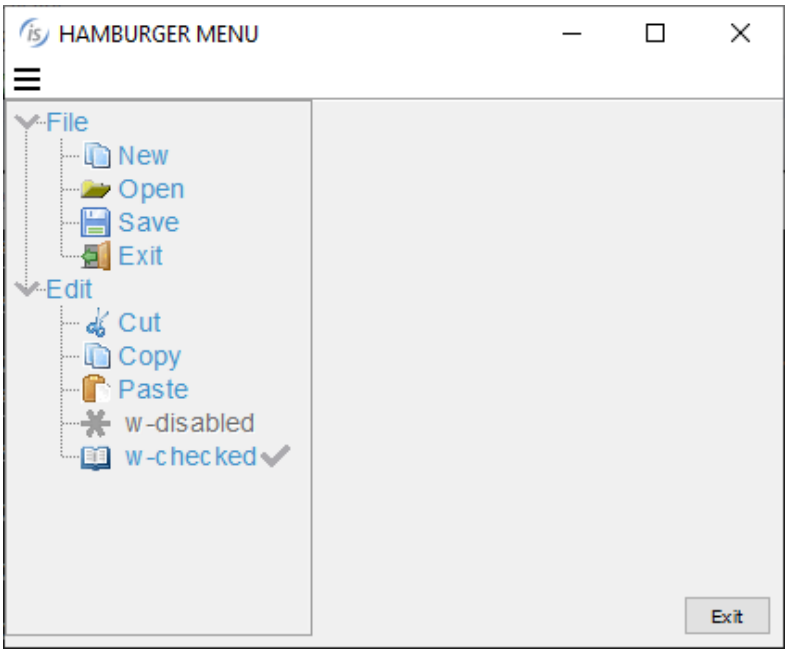
A new attribute called "menu-bar-flavor" has been implemented to easily change the default menu style of the application. Executing the code

```
call "W$MENU" using wmenu-set-attribute "menu-bar-flavor" "hamburger"
```

sets the hamburger menu as default menu type, and the entire application will use hamburger menus instead of the classic menu-bar.

Figure 9, *Hamburger menu,* shows a program running with a hamburger menu opened after pressing the hamburger button.

Figure 9. Hamburger menu



Glyph fonts

Glyph fonts are vector fonts that contain icons instead of characters, and have been widely used in web applications. Being vector-based make them especially suitable to scale without visual quality loss when used on high-DPI displays.

In the isCOBOL 2020 R2 release they are available for COBOL programs, using the new `"wbitmap-load-from-font"` op-code of the W$BITMAP library routine. Using this op-code, text and symbols can be converted to bitmaps, and can then be used on any control that supports `bitmap-handle` and `bitmap-number` properties, such as push-button, entry-field, grid, tree-view.

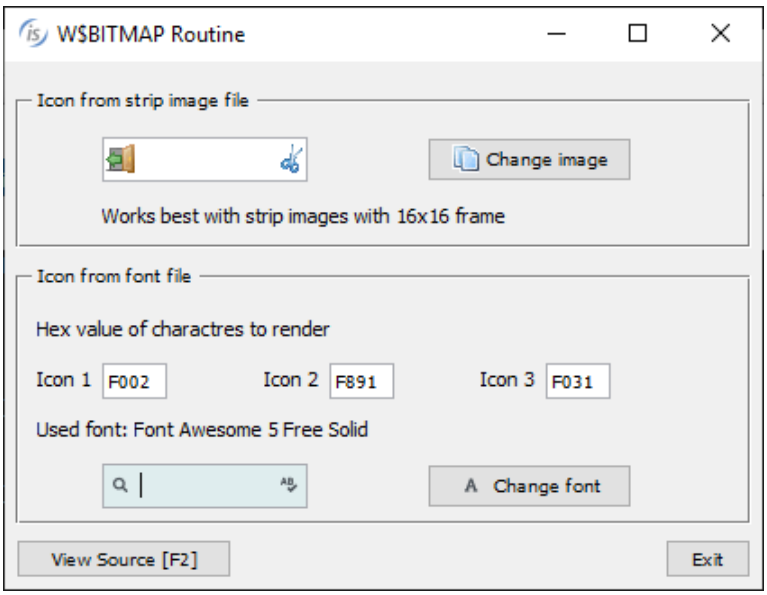
The following code snippet loads a font using the W$FONT routine, then it loads the glyph symbols of 3 characters identified by their Unicode numbers. Using the new op-code, the W$BITMAP routine can be called passing the font handle, the list of characters, the font-size and color to be used. The routine will generate a bitmap strip that can be used in controls that can handle bitmaps strips,

```
working-storage section.
77  h-font                  handle of font.
77  h-font-icon             pic s9(9) comp-4.
77  icon-size               pic 9(2).
77  icon-color              pic s9(9).
77  icon-characters         pic n any length.
...
screen section.
01  Mask.
...
    03 ef-font entry-field
       bitmap-handle h-font-icon bitmap-width 16
       bitmap-number 1  bitmap-trailing-number 2
       ...
    03 pb-font push-button title "Change font"
       bitmap-handle h-font-icon bitmap-width 16
       title-position 2 bitmap-number 3
       ...
procedure division.
    ...
    move "Font Awesome 5 Free Solid" to font-name
    call "w$createfont" using "Font Awesome 5 Free-Solid-900.otf" font-name
    initialize wfont-data
    set wfdevice-console to true
    move font-name       to wfont-name
    move 10              to wfont-size
```

```
CALL "W$FONT" using wfont-get-font h-font wfont-data
move nx"f002f891f031" to icon-characters
call "w$bitmap" using wbitmap-load-symbol-font h-font
                      icon-characters icon-size icon-color
                giving h-font-icon
display Mask
...
```

The final result is shown in Figure 10, *Controls with glyph symbols*, where the zoom icon is bitmap-number 1, and is loaded from the "Awesome 5 Free-Solid-900.otf" font, and has Unicode character nx"F002".

Figure 10. Controls with glyph symbols



This new feature makes it easy to enhance isCOBOL GUI applications to use glyph fonts as icons instead of loading bitmap images from disk.

Other GUI enhancements

The Grid control has been enhanced by adding a new `filter-types` property to better manage filter types for each column.

It's now possible to specify at a column level if the filter feature is available and the type of filter, as shown in the following code:

```
filter-types (0, 2, 0, 2, 1, 2, 2, 1)
```

A value of 0 will disable filtering for the column, 1 will remove the filter after a click on the red funnel icon, and a value of 2 will reopen the previously defined filter when clicking the yellow funnel icon.

The search panel has been enhanced by adding the ability to search with case insensitive rules. All buttons in the panel have now an icon.

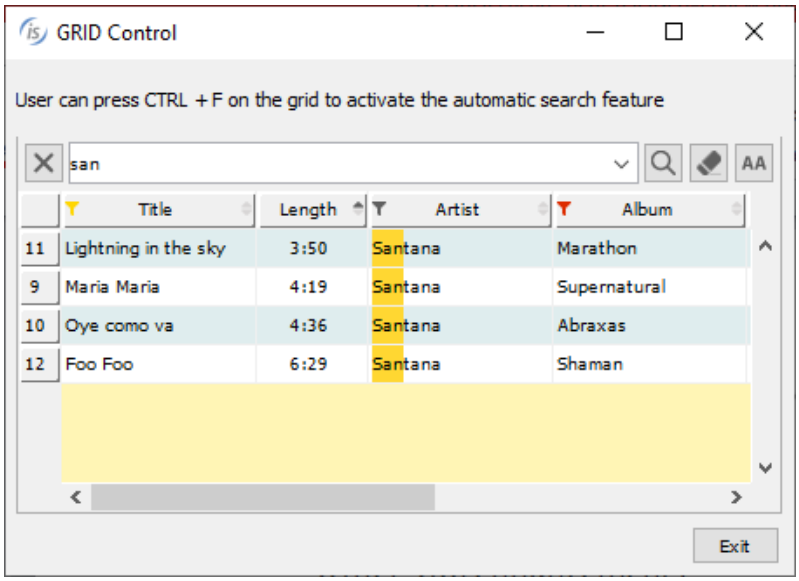
The search panel can now be opened programmatically using the new action-search value, as shown in the code below:

```
modify my-grid action action-search
```

The sort feature activated with sortable-columns or sort-types properties now have new icons. All grid icons can be customized by adding a JAR file that contains the image files to the CLASSPATH.

All these new grid features are shown in Figure 11, *Grid filter, sort and search features*.

Figure 11. Grid filter, sort and search features



The tree-view control now supports selecting multiple elements of the same level. This can be activated with the selection-mode property as shown in the code snippet:

```
selection-mode tvsm-multiple-interval-selection
```

With this feature enabled, multiple items can be selected using the usual Control and Alt key combinations.

Selected items can be inquired by the program using the items-selected property that returns the list of item handles:

```
inquire tv1 items-selected tv-items-sel
```

Figure 12, *Tree-View selection-mode,* shows the new tree-view feature in action.

Figure 12. Tree-View selection-mode

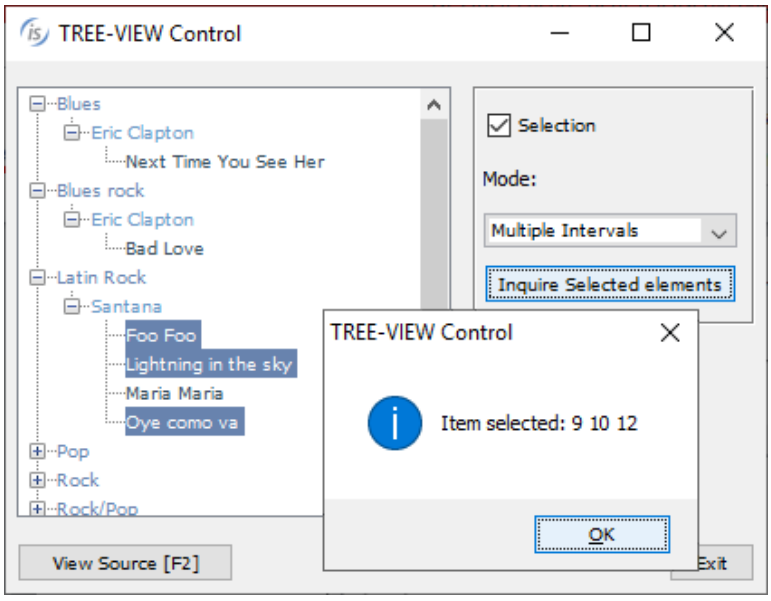


Figure 12 also shows the new message box, which has been revamped to provide a more modern look and feel. Additionally, the message box can now be centered in the monitor instead of the parent window using the CENTERED keyword, as shown in the following code snippet.

```
display message "Are you sure to continue?"
        title "Confirmation"
        type mb-yes-no
        icon mb-warning-icon
        default mb-no
        timeout 800
        centered
        giving mb-return
```

A new configuration property can be used to automatically apply the new CENTERED option at the application level:

```
iscobol.gui.messagebox.centered=true
```

It is also possible to provide a custom message box implementation using a new configuration property:

```
iscobol.gui.messagebox.custom_prog=PROGRAM_NAME
```

PROGRAM_NAME is a compiled COBOL program that is automatically called by the isCOBOL framework every time a display message statement is executed. The program receives the details of the message box as linkage section parameters as defined below, and can provide a customized implementation:

```
linkage section.
77  msgbox-text              pic x any length.
77  msgbox-title             pic x any length.
77  msgbox-type              pic 9.
77  msgbox-icon              pic 9.
77  msgbox-default           pic 9.
77  msgbox-timeout           pic 9(5).
77  msgbox-centered          pic 9.
procedure division using msgbox-text
                         msgbox-title
                         msgbox-type
                         msgbox-icon
                         msgbox-btn-default
                         msgbox-timeout
                         msgbox-centered
                         .
```

The web-browser control can now be used with the JxBrowser Java component, a powerful Chromium-based integrated browser for Java applications that processes and displays HTML5, CSS3, JavaScript, Flash etc. Customers that used JxBrowser in Java applications can now use it in isCOBOL applications by setting the following configuration:

```
iscobol.gui.webbrowser.class=com.iscobol.browser.jx.JXWebBrowser?licenseKey=...
```

isCOBOL Compiler

The isCOBOL 2020R2 compiler is faster in massive compilations, a new EasyLinkage feature has been implemented to easily convert C calls into pure Java libraries and additional syntax is supported to increase compatibility with other COBOL dialects.

Performance on massive compilations

The compiler has been reworked by removing the need for the tools.jar file to be in CLASSPATH, and the compilation of .java files to .class objects is now faster, especially when performing large command line compilations, such as when using the command:

```
iscc –options... source/*.cbl
```

Figure 13, *Comparison of compiler performance,* shows a performance comparison between isCOBOL 2019R1, 2019R2, 2020R1 and the current 2020R2. The tests were run using Windows 10 Pro 64-bit on an Intel Core i-7 Processor-8550U, 1.80 GHz with 16 GB of RAM, using Oracle JDK 1.8.0_251. All times are in seconds. The test was compiling a real application module consisting of 550 programs with more than 10,000 copy files, for a total of over 1 million lines of COBOL code. Different compiler options have been used depending on the type of compilation.

Figure 13. Comparison of compiler performance

type of compilation	2019R1	2019R2	2020R1	2020R2
release mode -ostrip	641	536	522	430
debug mode -d	784	706	675	585
debug mode -dx	1349	1083	1082	834
debug mode -dx and .list with -lf	1570	1190	1181	878

isCOBOL is constantly being improved for performance and customers using older versions now have another good reason to upgrade their release to take advantage of faster compiling times.

EasyLinkage for Java migration

When migrating COBOL code that relies on C code where COBOL code makes native function calls using the CALL statement, it is usually advised that you migrate the C code to Java to have more readable code and avoid depending on C. To help in the process, the new isCOBOL 2020 R2 compiler introduces a new feature that generates Java stub sources to handle COBOL parameters received in CALL statements.

Assume, for example, the following COBOL code, which uses a C library with 3 different functions that are being ported to isCOBOL and the C library is being migrated to Java:

```
WORKING-STORAGE SECTION.
77  w-path        pic x(256).
77  path-len      pic 9(3).
01  w-exec.
    03 w-command  pic x(128).
    03 w-options  pic x(64).
    03 w-param    pic 9(2) comp.
...
PROCEDURE DIVISION.
MAIN.
 ...
 call "MyLibrary"
 call "func_initialize"
 call "func_get_path" using by value path-len
                      by reference w-path
 call "func_exec" using w-exec
 ...
```

The source code can be compiled using the new configuration setting:

```
iscobol.compiler.easylinkage=2
```

to automatically generate the stub sources called: MYLIBRARY.java, FUNC_INITIALIZE.java, FUNC_GET_PATH.java and FUNC_EXEC.java. The main problem when integrating COBOL code with another language is understanding how to code the proper memory structures to receive the COBOL parameters: how are group variables handled? How is a comp variable stored?

When compiling with the above property set, the isCOBOL compiler takes care of this complexity by generating the correct parameter definitions in the Java source code. Developers only need to focus on porting the C logic to Java. The EasyLinkage feature even handles C functions with variable numbers of arguments and different sizes.

For example, in the FUNC_EXEC.java source the COBOL group level w-exec has been declared as:

```
//  variable declaration W-EXEC
   public byte wExec$0[];
   public    com.iscobol.types.PicX wExec;
   public    com.iscobol.types.PicX wCommand;
   public    com.iscobol.types.PicX wOptions;
   public    com.iscobol.types.NumericVar wParam;
    ...
    {
      wExec$0=Factory.getMem(194);
      wExec=Factory.getVarAlphanum(wExec$0,0,194,...,"W-EXEC",...);
      wCommand=Factory.getVarAlphanum(wExec,0,128,...,"W-COMMAND",);
      wOptions=Factory.getVarAlphanum(wExec,128,64,...,"W-OPTIONS",...);
      wParam=Factory.getVarBinary(wExec,192,2,...,"W-PARAM",...);
    }
    ...
/*  Write the routine logic here
    ...
```

Additionally, the compiler generates by default the source files in specific subfolders called:

easydb for database bridge generation (any kind of database option)

easylinkage for EasyLinkage generation (both link and stub)

servicebridge for web service bridge generation (both REST and SOAP)

bean for web service bean generation

By default, these folders are generated inside the sources folder, but can be configured using the following compiler property:

iscobol.compiler.generate.root_dir=path

The .class files created by compiling the generated COBOL sources (easydb, servicebridge and bean) are by default located in the same folder where the main COBOL program .class is created, in the path specified using the –od compiler option. It is also possible to create the same subfolder names under the main class folder by using: **iscobol.compiler.generate.keep_structure=true,** resulting in a clearer organization of folders.

Compatibility

The compiler now supports the EXHIBIT statement when using the –cv option for IBM COBOL compatibility. Running the code below

```
MOVE "ABC" TO MY-VAR
EXHIBIT NAMED MY-VAR
```

when compiled using the –cv option produces the following output on sysout when run:

```
MY-VAR=ABC
```

This is similar to the DISPLAY UPON SYSOUT statement, but it's now fully supported and doesn't require code to be manually changed when porting to isCOBOL.

The ESQL DECLARE CURSOR statement has been enhanced to support the SENSITIVE and INSENSITIVE clauses. Code such as:

```
exec sql
     declare curs1 sensitive cursor for
             select * from companies
end-exec.
exec sql
     declare curs2 insensitive cursor for
             select * from companies
end-exec.
```

are now supported and can be compiled to simplify the migration from other COBOL ESQL Pre-compilers.

Insensitive allows the cursor to move forward and backward through the data, changes made while the cursor is open are ignored. *Sensitive* allows the cursor to move forward and backwards through the data, changes made while the cursor is open are immediately available.

This behavior is similar to the runtime configuration `iscobol.jdbc.cursor.type`, and both can be used in the same program, allowing additional flexibility.

isCOBOL Runtime

Starting from isCOBOL 2020R2, new configuration settings and new library routines are available to customize behavior and improve compatibility.

The new configuration settings are:

`iscobool.key.<keystroke>=search=<context>` to customize the search action in a specific component. By default, pressing Ctrl+F on grid, print-preview and web-browser makes the Search panel appear. The key combination can be customized: for example it can be set to Shift+F4 by using the following configuration settings:

```
iscobol.key.*f=search=
iscobol.key.^f4=search=grid,print-preview,web-browser
```

If you need to disable the Ctrl+F feature for the web-browser only, the following setting can be used:

```
iscobol.key.*f=search=print-preview,grid
```

`iscobol.call_cancel.hook=className` to specify a hook class for CALL and CANCEL statements. The class can be written in pure Java or in COBOL using the appropriate syntax CLASS-ID to implement the interface "com.iscobol.rts.CallHandler". The list of required methods that need to be defined for this interface are:

```
public void afterCall(IscobolCall call, Object[] argv);
public void afterCancel(IscobolCall call);
public void afterCancelAll();
public void beforeCall(IscobolCall call, Object[] argv);
public boolean beforeCancel(IscobolCall call);
public boolean beforeCancelAll();
```

`iscobol.jdbc.auto_connect=true` to automatically connect to the database without using an explicit ESQL CONNECT statement in the program. When the first ESQL statement is executed, the connection is automatically established.

`iscobol.jdbc.user=userName` to supply a username credential without specifying it in the iscobol.jdbc.url property. This can be used also in conjunction with `iscobol.jdbc.auto_connect=true`

iscobol.jdbc.password=pwd to supply the connection password without specifying it in the iscobol.jdbc.url property. This can be used also in conjunction with `iscobol.jdbc.auto_connect=true`

New library routines have been implemented to enhance the compatibility with the MicroFocus® dialect:

CBL_ALLOC_MEM to allocate memory. The allocated memory amount is expressed in bytes, and it returns a pointer used to release allocated memory.

CBL_FREE_MEM to release the previously allocated memory.

CBL_GET_CURRENT_DIR to retrieve the current folder name.

CBL_READ_DIR to retrieve the current folder absolute path name.

The following is a code snippet of a migrated COBOL program that calls all the new supported library routines:

```
WORKING-STORAGE SECTION.
77  path-name          pic x(256).
77  path-name-length   pic x comp-x.
77  dir-status-code    pic x(2) comp-5.
77  flags              pic x(4) comp-5.
77  name-length        pic x(4) comp-5.
77  mem-pointer        usage pointer.
77  mem-size           pic x(4) comp-5.
77  mem-flags          pic x(4) comp-5.
77  mem-status-code    pic x(2) comp-5.
...
PROCEDURE DIVISION.
...
    call "CBL_READ_DIR" using path-name
                              path-name-length
                      returning dir-status-code
    call "CBL_GET_CURRENT_DIR" using by value     flags
                                     by value     name-length
                                     by reference path-name
                                     returning    dir-status-code
    call "CBL_ALLOC_MEM" using          mem-pointer
                               by value mem-size
                               by value mem-flags
                              returning mem-status-code
    call "CBL_FREE_MEM"  using by value mem-pointer
                              returning mem-status-code
```

isCOBOL Debugger

Starting from isCOBOL 2020R2, the isCOBOL Debugger has been enhanced and can now debug programs without having access to source files. When compiled with the -d or -dx options, the compiler now embeds an encrypted copy of the source code in the compiled .class file. Such a .class file can now be fully debugged anywhere. This is completely transparent for developers, who will have access to the COBOL source during a debugging session as they had previously, and can take advantage of the same debugger commands that need references to source code lines, such:

```
br 123 MYPROG.cbl
```

to set a breakpoint on line 123 of MYPROG.cbl program.

This is a major improvement that enhances the usability of the isCOBOL Debugger in any scenario, from stand-alone debugging (iscrun –d PROGNAME), thin-client debugging (iscclient –d –hostname ip –port n PROGNAME), and remote debugging for all other architectures (iscrun –d –r hostname port)

The Debugger can still load source files from disk if the .class programs are compiled with a previous compiler release for backward compatibility.

In addition, Debugger now has a new integrated View named Console in the isCOBOL IDE that has the ability to attach and de-attach the standard sysout and syserr in its view. When the Console is attached, all the sysout and the syserr output is shown in the new View, using black and red colors respectively. When the Console is de-attached, all the sysout and syserr are shown back in the system console. This results in a clearer view, where the Output View is on the bottom-left corner and shows the results of commands executed from the developer and the single line of code that is executed with “step” command, and the Console view, when attached to activate, is shown in the bottom-right corner and shows the sysout/syserr produced by COBOL program or by the isCOBOL runtime error handling.

In Figure 14, *Debugger Console view*, the bottom-right portion of the Console view shows the attached sysout/syserr. A pop-up menu is also available to Copy, Select and Clean the content. Figure 15, *Debugger Console settings*, shows the dialog that appears when selecting the Window menu item Settings – Console, that allows redirection of sysout/syserr which is especially useful in Thin Client, where this change is not dynamic.

Figure 14. Debugger Console view

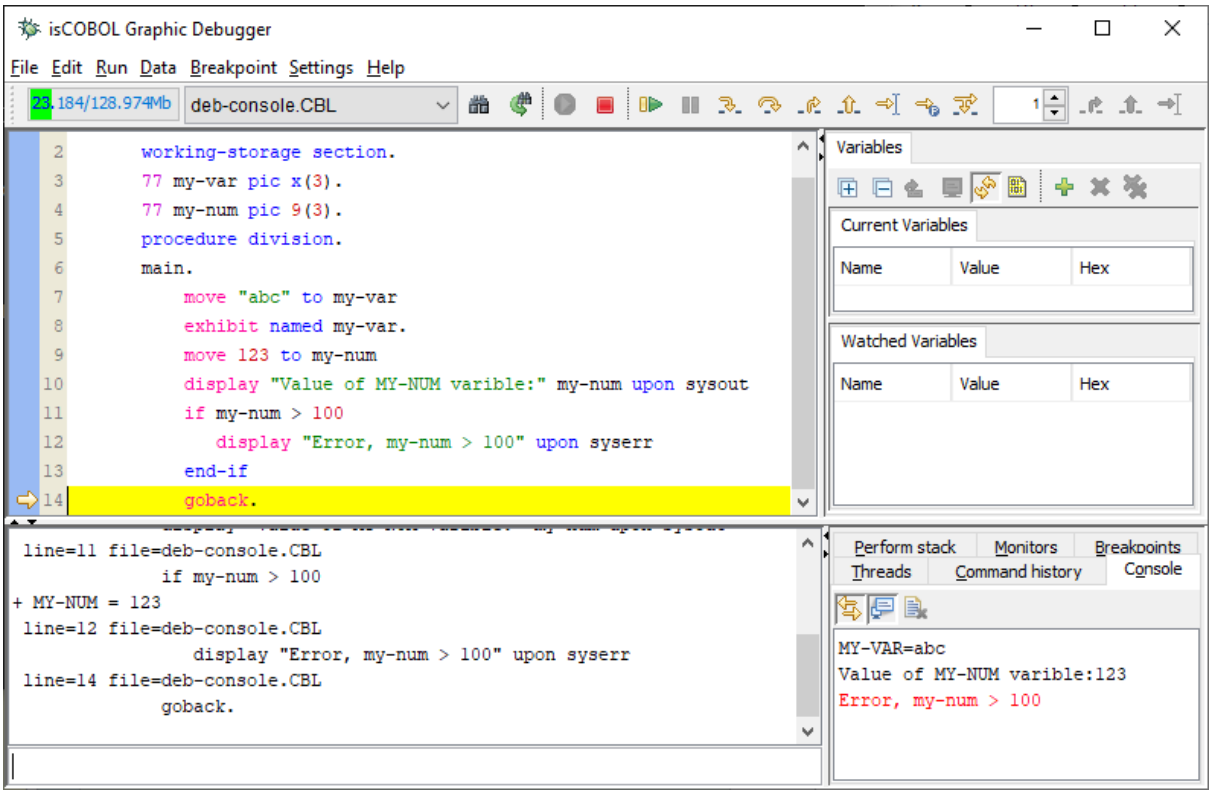
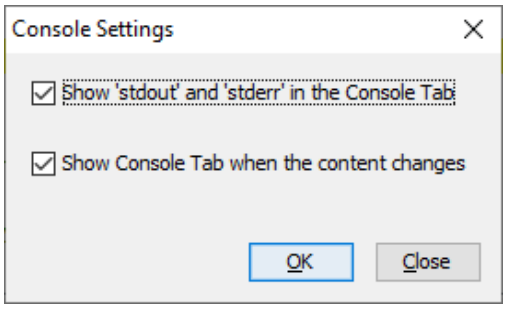


Figure 15. Debugger Console settings



c-treeRTG v11.9

A new version of c-treeRTG. V11.9, is available and embedded in the Veryant 2020R2 release. Version v11.9 contains many new features such as TLS/SSL security communication between client and server and other improvements.

Additional details are explained in the installed documentation located in the "c-treeRTG v11.9.0\Server\V11.9.0_Veryant_Delivery.pdf" folder.

TLS/SSL

SSL communication is enabled with a new server configuration setting in the ctsrvr.cfg configuration file, as shown below:

```
SUBSYSTEM COMM_PROTOCOL SSL {
       SERVER_CERTIFICATE_FILE ctree_ssl.pem
       SSL_CONNECTIONS_ONLY YES
}
```

The settings allow the definition of the SSL certificate file, and whether an SSL connection is mandatory (default setting is NO). Additional configuration settings are optionally available.

If a client is configured with a standard c-tree xml client configuration file, the new attributes "ssl" and "sslcert" in the `<instance>` configuration element must be set, as shown below:

```
<config>
   <instance ssl="yes" sslcert="ctsrvr.pem" server="FAIRCOMS@hostname"
             user="admin" password="ADMIN"
             connect="yes" versioncheck="no">
   ...
   </instance>
</config>
```

If the client is configured using an isCOBOL properties file, the new settings can be configured as shown below:

```
iscobol.file.index.server=FAIRCOMS@hostname
iscobol.file.index.user=admin
iscobol.file.index.password=ADMIN
iscobol.file.index.ssl=true
iscobol.file.index.sslcert=/path/filename
```

Other improvements

Additional c-tree client configuration properties have been implemented, such as:

iscobol.file.index.prefetch.ttl=n to specify how long pre-fetched records should be kept, in seconds

iscobol.file.index.keycompress.rle=true to enable RLE compression on keys

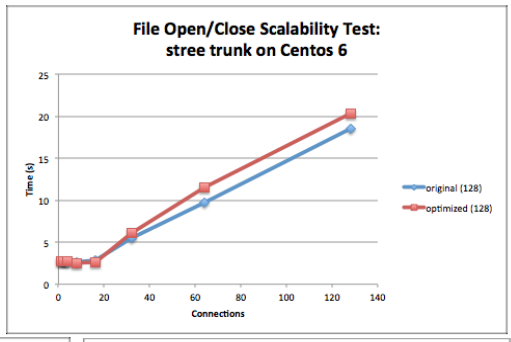
iscobol.file.index.keycompress.vlennod=false to restore the legacy LEADING compression on keys

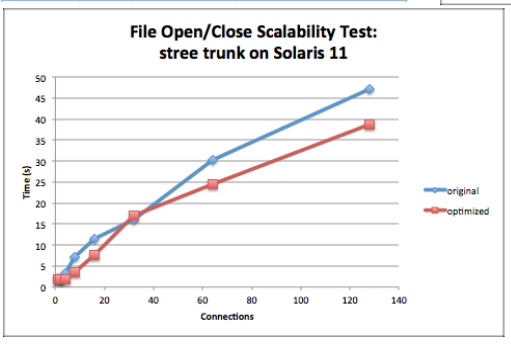
iscobol.file.index.memoryfile.persist=false to unload memory files when the run unit terminates, instead of when the c-treeRTG server shuts down.

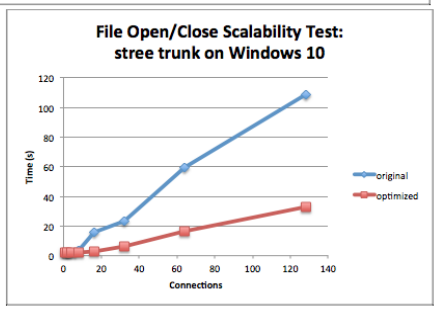
The performance of c-treeRTG Server, when handling concurrent file open and close operations, has been improved, especially when scaling to systems with a large number of CPU cores and concurrent database connections. Figure 16, *c-treeRTG scaling test* shows scaling test results, with the number of files ranging between 1 and 128, running 2,000 iterations and 128 connections. Result times are in seconds.

Figure 16. c-treeRTG scaling test

c-tree	Files	Windows 10	Solaris 11	Centos6
original	1	2.022	2.837	3.067
	2	2.231	3.654	2.902
	4	2.558	14.189	2.915
	8	6.439	11.581	2.698
	16	18.607	22.002	3.139
	32	27.337	32.489	6.589
	64	60.170	50.911	11.589
	128	113.519	78.803	21.287
optimized	1	1.804	1.894	2.713
	2	1.795	1.771	2.687
	4	1.940	2.149	2.597
	8	2.036	2.706	2.575
	16	2.357	3.686	2.492
	32	5.241	7.010	5.355
	64	12.379	13.462	10.452
	128	25.849	27.322	18.778

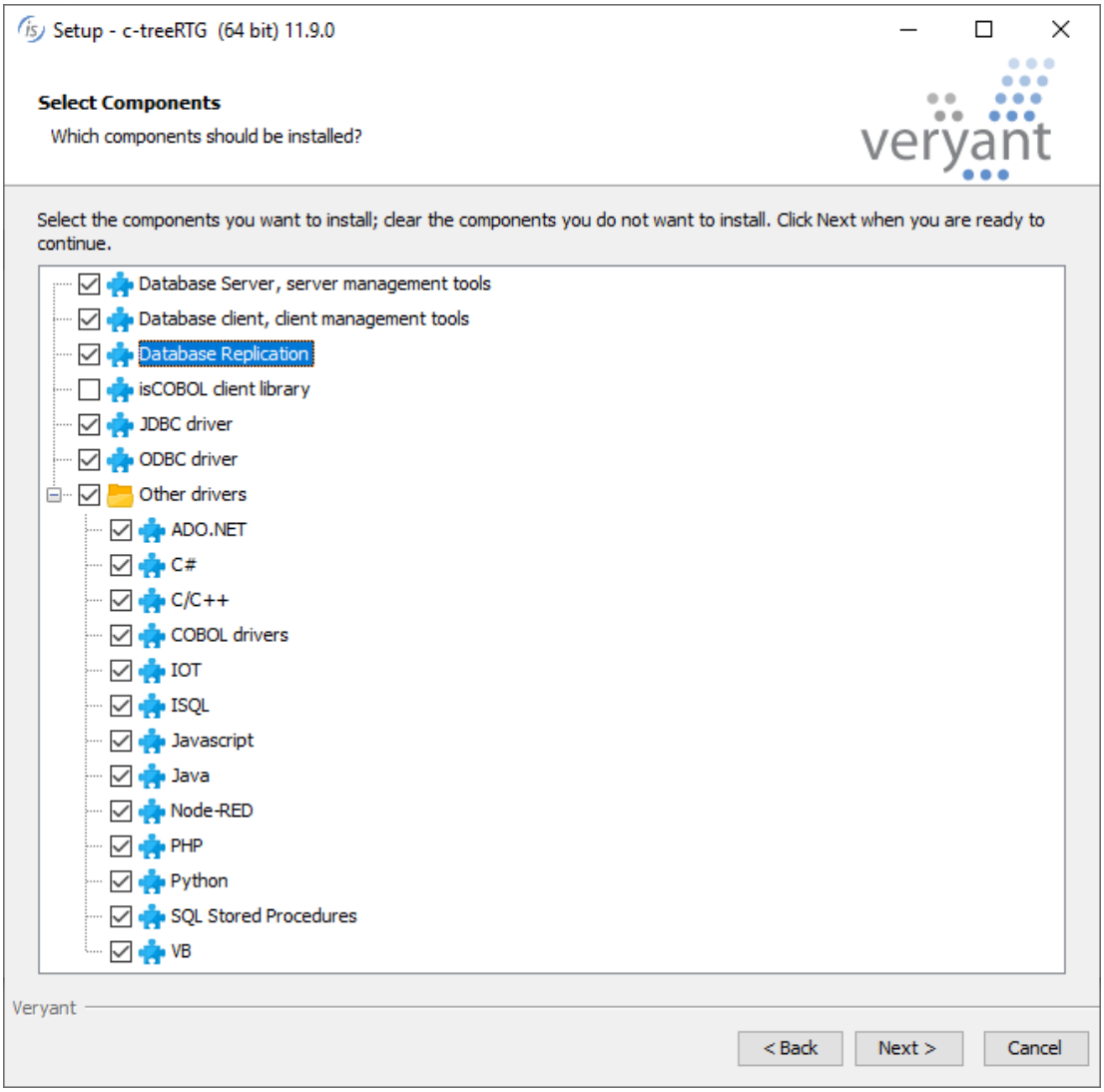






The new c-treeRTG Veryant setup, as shown In Figure 16, *c-treeRTG 11.9 setup,* contains additional components, such as the Replication agent, making it simpler to install, and provides new Drivers with tutorials for additional languages, including C#, C/C++, IOT, ISQL, Javascript, Node-red and VB. This enhances the interoperability of languages that share the same data. To enable these features, you need to replace the embedded license, which only allows access from isCOBOL, with a full license that allow full SQL access and enables the Replication feature.

Figure 17. c-treeRTG 11.9 setup



Additional improvements

isCOBOL compiler can now be integrated in Apache Ant builds. Ant is a software tool for automating software build processes implemented using Java and is well integrated in software for automation servers like Jenkins. These automation servers help automate the parts of software development related to building, testing, and deploying. Products like Jenkins and Ant are often used together to facilitate continuous integration and continuous delivery (CI/CD).

An integrated system to use isCOBOL from Apache Ant is now available to simplify the usage of isCOBOL compiler in Ant build script.

To use isCOBOL inside Ant script you can now add the following tags to the build configuration file:

```
<taskdef name="iscc" classname="com.iscobol.ant.iscc"/>
```

or use the syntax:

```
<project name="HelloWorld" default="build" basedir="."
         xmlns:veryant="antlib:com.iscobol.ant">
```

The task name “iscc” provides a task to allow compilation, using the syntax shown below:

```
<!-- Cobol sources -->
<property name="src.dir" location="." />
<!-- Java classes -->
<property name="build.dir" location="prg" />
...
<iscc
    javacOptions="-classpath ${iscobol-classpath-prop} "
    nosummary="false"
    nowarn="true"
    noerr="true"
    force="true"
    options="-sp=../isdef"
    failOnError="true"
    destDir="${build.dir}">
    <fileset dir="${src.dir}">
       <include name="**/*.cbl"/>
    </fileset>
</iscc>
```

Files with a .cbl extension in the current folder and its subfolders are compiled, and the .class will be generated in the “prg” destination folder.

These are the details of supported options in the iscc Ant task:

`javac=path` to specify the external javac compiler to be used, default is none

`javacOptions=options` to pass options to the java complier, default is none

`nosummary=false` to suppress the display of compiler summary information, default is true

`nowarn=true` to enable or disable compiler warnings, default is false

`noerr=true` to enable or disable compiler errors, default is false

`force=true` to force compilation a COBOL program even if it is not out of date, default is false

`options=options` to set the isCOBOL compiler options used by the iscc command, default is -jc

`failOnError=false` to continue the build process even if the compiler reports a severe error, default is true

`destDir=path` to specify the name of the folder where the compiler output is to be written, default is none

`fileset=fileset` to set the standard Ant fileset, defining a group of COBOL sources to be compiled. For details on the syntax see: https://ant.apache.org/manual/Types/fileset.html

Database Bridge

The EDBI routines generated by the Database Bridge functionality when targeting PostgreSQL are now certified for PostgreSQL release 12.

EIS improvements

A new overload method in HTTPHandler class has been added to allow setting the generation of a DummyRoot parameter:

```
HTTPHandler:>displayEx(content, hasDummyRoot)
```

The DummyRoot is an automatically generated root element added during the creation of the XML or JSON representation of a working storage item, if needed.

isCOBOL utilities improvements

The isCOBOL Server Panel now supports the Threads View and Thread stack even when the Application Server is executed with a JRE.

The graphical ISMIGRATE wizard can now read isCOBOL configuration files when it starts. This allows you to preload settings to be used in the wizard by using the iscobol.ismigrate.* and other relevant properties in configuration files, and referencing the configuration file when starting the wizard. Previously, these variables could only be used when running ISMIGRATE from the command line.

ISMIGRATE has also been expanded to extend existing files. This feature can be enabled from the command line by setting the configuration property:

```
iscobol.ismigrate_open_extend=true
```