isCOBOL™ Evolve

isCOBOL Evolve 2020 Release 1 Overview

isCOBOL Evolve 2020 Release 1 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL™ Evolve, isCOBOL Evolve 2020 R1.

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

isCOBOL Code Coverage and isCOBOL Unit Test are enterprise level features that enable developers to write robust test suites and check their effectiveness, allowing the production of a more stable code base in applications.

The 2020R1 release has many new features for GUIs, such as a new control: scroll-pane, and the table-view style for tree-views. These are features that let developers modernize applications.

The EIS suite has been upgraded, enhancing both WebClient and WebDirect with new features.

Details on these enhancements and updates are included below.

isCOBOL Code Coverage and isCOBOL Unit Test features

Starting with the 2020R1 Release, isCOBOL is introducing features with enterprise users in mind: isCOBOL Code Coverage and isCOBOL Unit Test. These features have been available in other languages, such as Java, and now they're available to isCOBOL developers as well. They will help developers produce better quality tests for COBOL applications.

These features are now available in the isCOBOL Evolve suite, and can be accessed from the command line or from the Eclipse-based isCOBOL IDE.

isCOBOL Code Coverage will measure the degree to which the source code of a program is executed during test suite runs. A program with high code coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low code coverage. This feature can also be used without a real test suite, and it can be enabled via a runtime option.

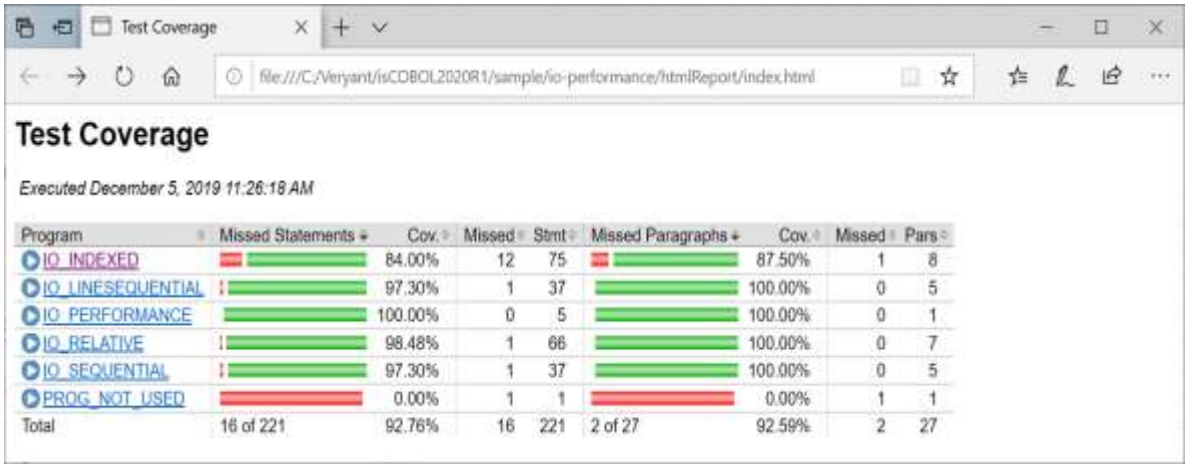
When running the command:

```
iscrun –coverage IO_PERFORMANCE
```

isCOBOL Code Coverage creates a folder called "htmlReport" which contains an HTML report of the code analysis. Figure 1, *isCOBOL Code Coverage global report,* shows the report with a list of all programs executed and the percentage of total and individual program code coverage.

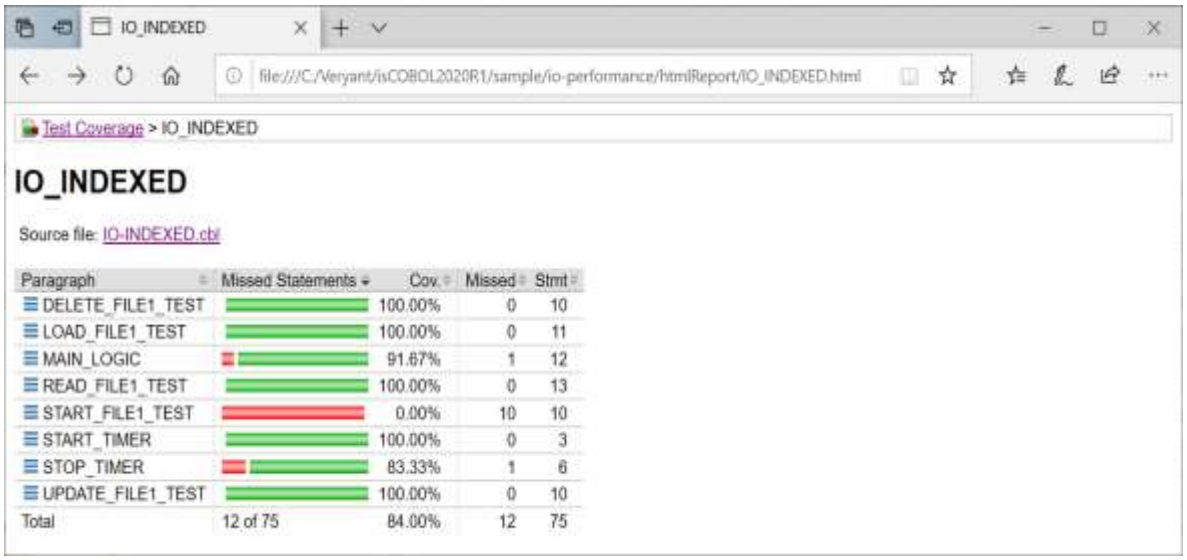
The report shows both the percentage and number of missed statements and paragraphs in each program.

Figure 1. isCOBOL Code Coverage global report



Clicking on the specific program name opens the report related to the single program, highlighting all COBOL paragraphs with the specific percentage of code coverage, as shown in Figure 2, *isCOBOL Code Coverage program report*.

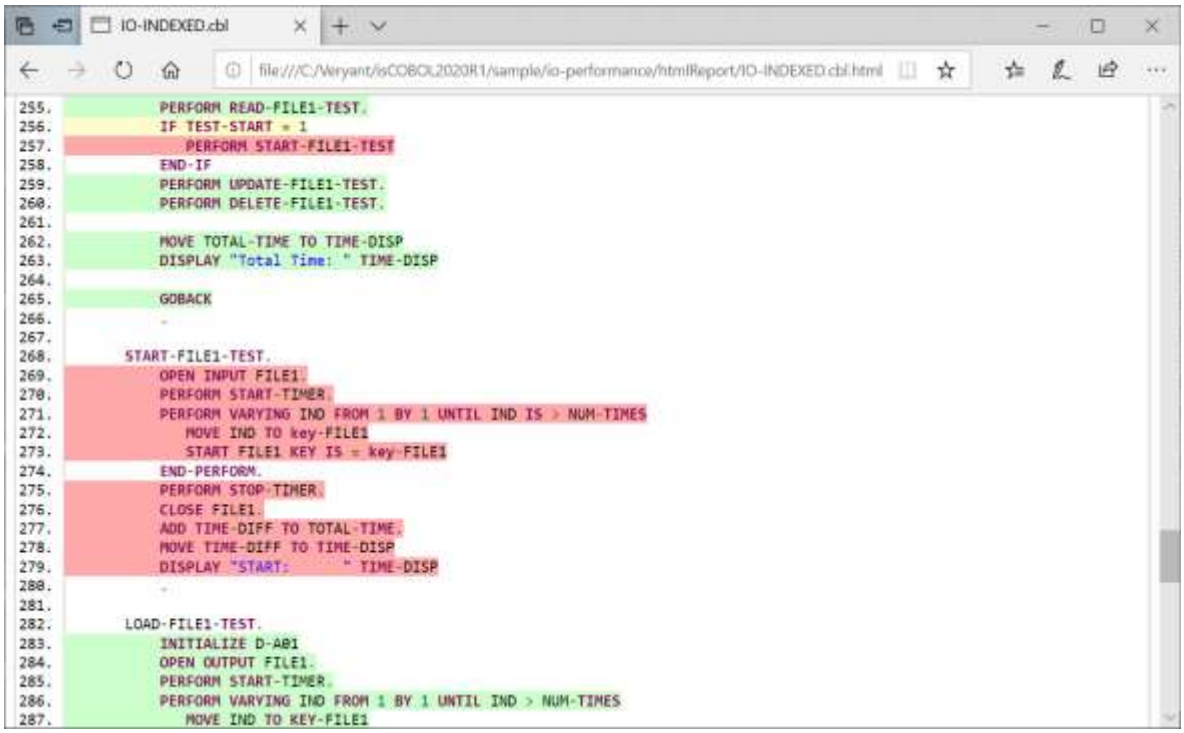
Figure 2. isCOBOL Code Coverage program report



Clicking on the source file name shows the corresponding source code, with a green background showing executed code, and a red background showing non-executed statements. A yellow background shows executed conditional statements. All colors are Eclipse's standard for code coverage available for the Java language.

In Figure 3, *isCOBOL Code Coverage source code*, it's easy to understand that the paragraph START-FILE1-TEST was never executed because the IF condition was never satisfied. With this information in mind, better tests can be developed to cover all of the code base.

Figure 3. isCOBOL Code Coverage source code



New configuration settings allow you to customize the report generated by isCOBOL Code Coverage:

- `iscobol.coverage.sessionname=name` sets the name of the coverage session; the default value is the name of the main program
- `iscobol.coverage.sourcefiles=path` sets the list of paths where the COBOL source files are located. If not set, the current folder will be used.
- `iscobol.coverage.classfiles=path` sets the list of paths where the class files are located, and are used by isCOBOL Code Coverage to build the report. If not set, the report will be built using the loaded isCOBOL classes
- `iscobol.coverage.html=path` sets the directory in which isCOBOL Code Coverage will create the report. If not set, the default value is "./htmlReport"

- **iscobol.coverage.xml=path** sets the file path of xml report. If not set, the xml report is not generated. This is needed when using isCOBOL Code Coverage inside the IDE to take advantage of its specific View.
- **iscobol.coverage.analysis.excludes=path** sets the list of isCOBOL classes that will be excluded from the analysis. It can contain the wildcard '*', for example A*
- **iscobol.coverage.analysis.includes=path** sets the list of isCOBOL classes that will be included in the analysis. It can contain the wildcard '*', for example B*

isCOBOL Unit Test lets developers create automated test suites, which are designed to test that sections of code execute as intended. The more comprehensive the tests included in a suite are, the more stable the resulting application will be. The goal of unit testing is to isolate sections of a program and ensure that they are working correctly.

This feature is activated using a command line option, such as:

```
iscrun -iut -J-ea
```

The –ea Java option is needed to take advantage of the ASSERT statement used in the test programs to show the reason of the failure in the isCOBOL Unit Test report. For example, the following assert statement checks that the condition of a variable named string1 is equal to “my string”. If the check fails, the string in the “otherwise” clause will be added in the report:

```
assert string1 = "my string"
         otherwise "Test string manipulation: Error"
```

If the assert condition is true, the program continues to the next statement. If the entire program is executed without any assert statements failing and no exceptions are raised, it means that the test is successful.

The only mandatory configuration option to set is:

iscobol.unit_test.list_file=path

where path is a single file path or a list of paths that define the list of programs to be executed.

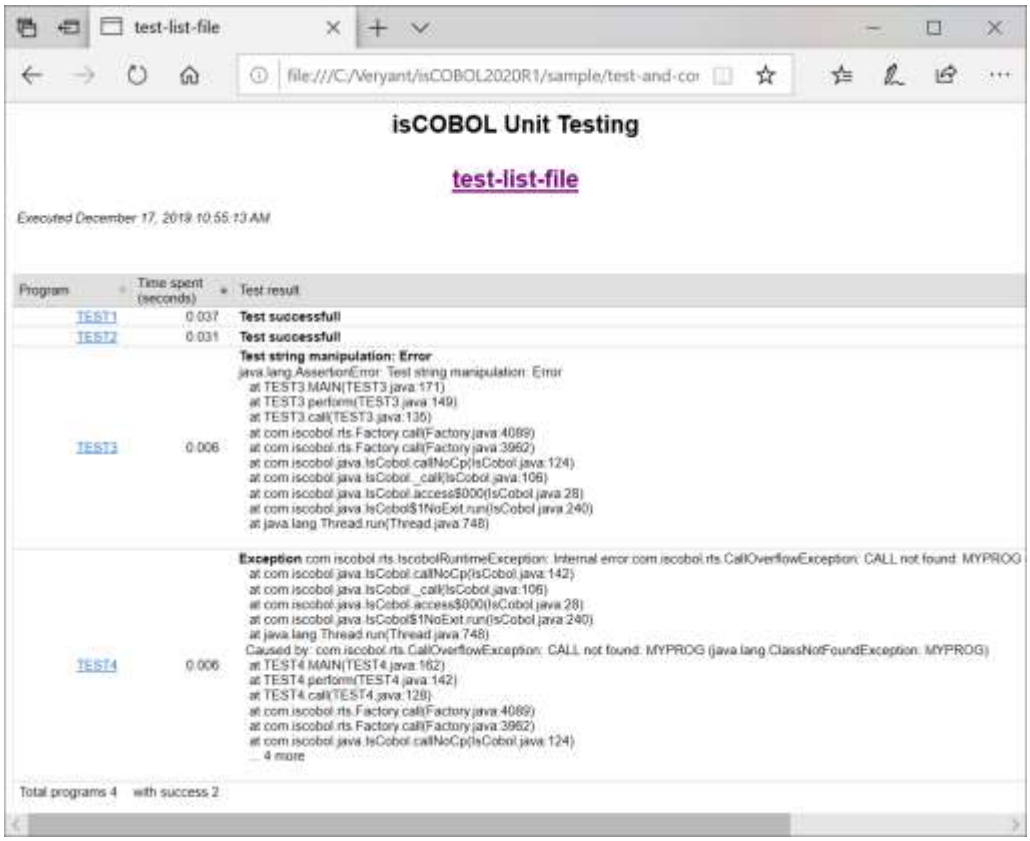
For example, if the test suite needs to execute four programs called TEST1, TEST2, TEST3 and TEST4, the file declared in the iscobol.unit_test.list_file needs to contain:

```
TEST1
TEST2
TEST3
TEST4
```

Figure 4, *isCOBOL Unit Test,* shows the html report created after running the test suite. In this example, the first 2 programs are executed correctly while TEST3 fails because the above ASSERT statement is executed and the condition is evaluated to false. TEST4 fails because a runtime exception has been raised.

The report will show a list of executed programs, the execution time, and if the tests completed with success. If not, the failed assertions or runtime errors will be included.

Figure 4. isCOBOL Unit Test



The Unit Test reports con be customized using the following configuration properties:

- `iscobol.unit_test.html=name` sets the directory in which the Unit Test will create the report. Its default value is "./htmlReport"
- `iscobol.unit_test.xml=path` sets the file path of xml report. If not set, the xml report is not generated. This is necessary when using the Unit Test feature inside IDE to take advantage of its specific View

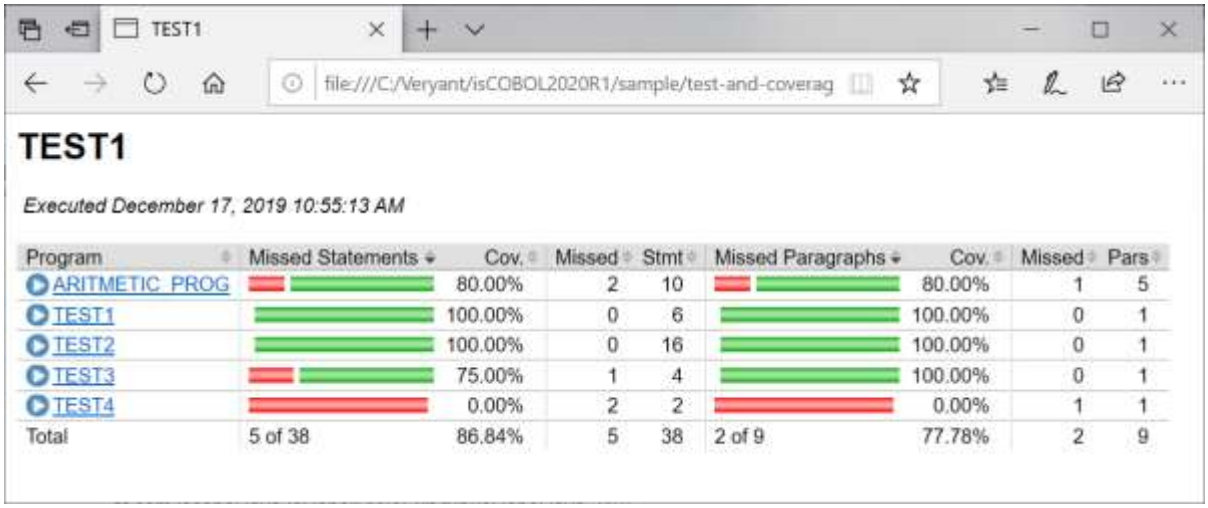
It's useful to run isCOBOL Unit Test in conjunction with isCOBOL Code Coverage, and both can be activated from the command line:

```
iscrun -coverage -iut -J-ea
```

This can help identify which portions of code still don't have tests associated with them, to make sure the test suites are complete enough to cover all the application code.

In this scenario the isCOBOL Unit Test report contains a test-list-file link to isCOBOL Code Coverage reports, as shown in Figure 5, *isCOBOL Code Coverage report from isCOBOL Unit Test*.

Figure 5. isCOBOL Code Coverage report from isCOBOL Unit Test

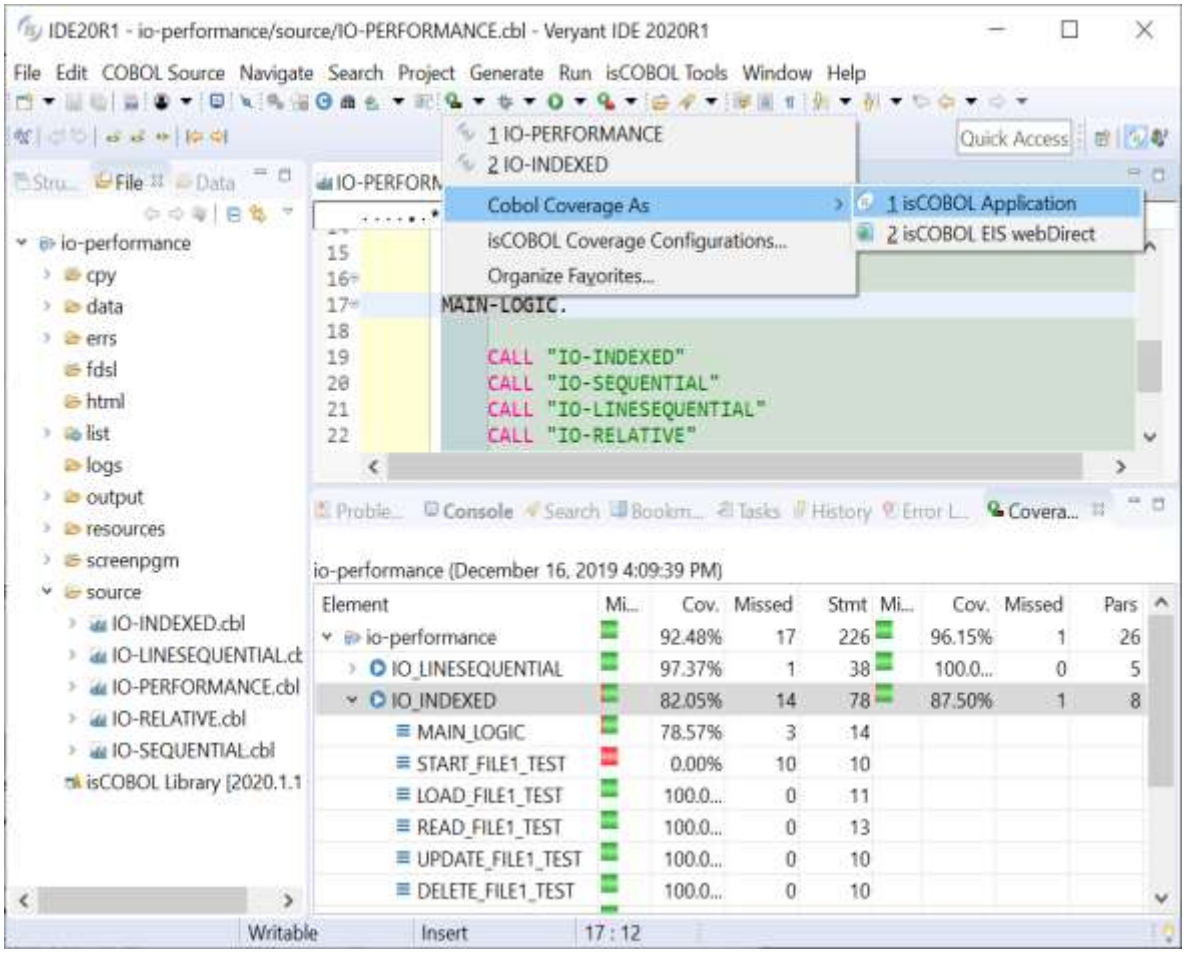


isCOBOL IDE enhancements

In isCOBOL IDE 2020 R1 you can also execute the new Enterprise features in an integrated system, activating dedicated Views to show and analyze the results of isCOBOL Code Coverage and isCOBOL Unit Test.

To execute the isCOBOL Code Coverage feature, as shown in Figure 6, *isCOBOL Code Coverage in isCOBOL IDE*, the toolbar button "Cobol Coverage As" can be used, and the result of the analysis is shown in the Coverage view. The same menu item is available under the main menu Run and in the source file's context menu.

Figure 6. isCOBOL Code Coverage in isCOBOL IDE

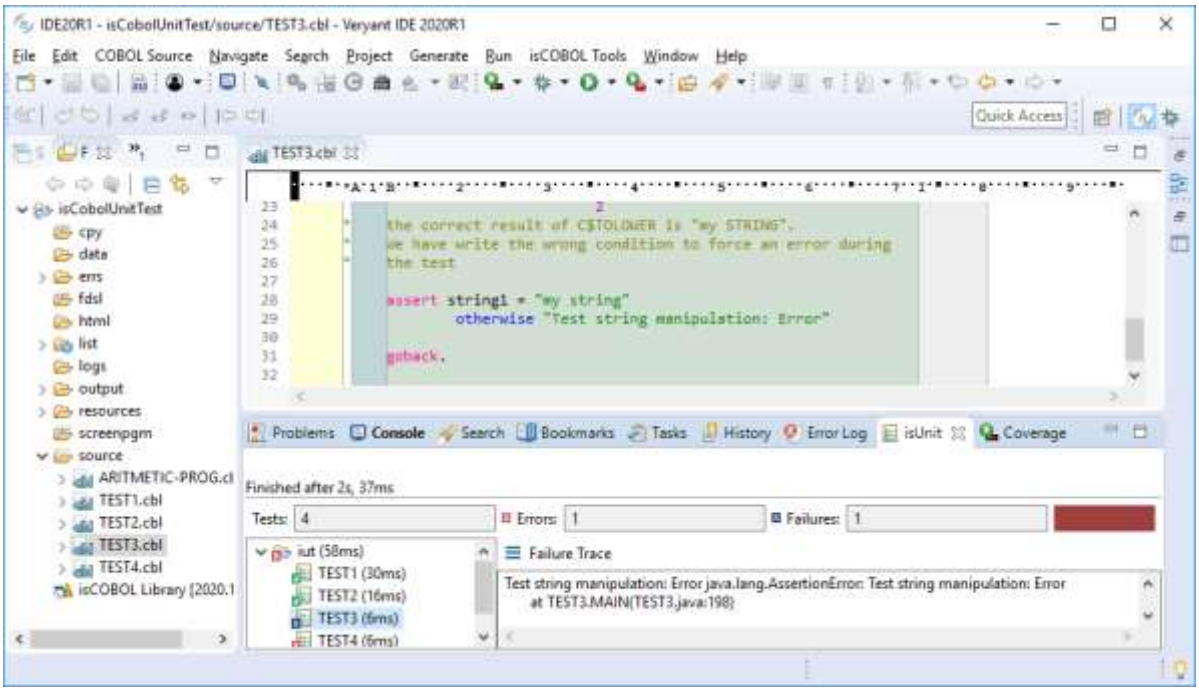


isCOBOL Unit Test feature can also be accessed from the isCOBOL IDE via a specific run, debug or coverage configuration option. When test execution is completed a specific view will be shown, the isUnit view, as depicted in Figure 7. *isCOBOL Unit Test in the IDE.* This view shows the list of executed programs, the total number of errors (exceptions) that occurred, the number of failed ASSERT statements and a color bar showing if all tests passed or one or more failed.

Clicking a program in the list displays details relative to that program.

The view is inspired by the standard Eclipse Unit Test view available for the Java language.

Figure 7. isCOBOL Unit Test in the IDE

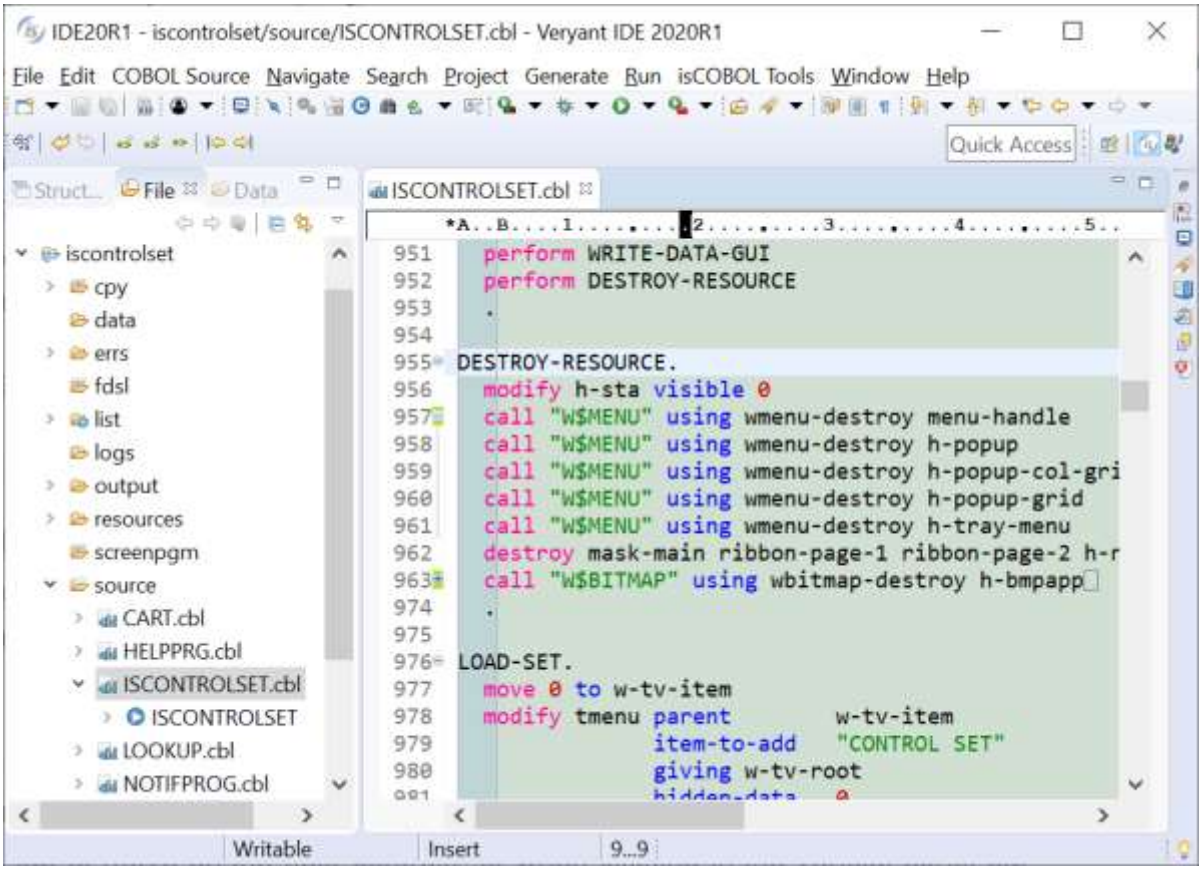


The isCOBOL IDE's Code Editor now supports customized code folding. This feature is useful when working with a large source code file, and can help developers hide less relevant pieces of code while concentrating on the important parts.

To enable this feature, the Enable folding option must be set in the Preferences / isCOBOL / Editor configuration section, which activates standard folding on sections and paragraphs. To activate a custom folding, select the relevant lines, right-click the folding bar and select the option "Add custom folding". Custom folding sets can be added and removed as needed.

Figure 8, *IDE custom folding*, shows standard folding and two custom folding sets, the first one of which is opened and the second is closed.

Figure 8. IDE custom folding



GUI enhancements

Many improvements to GUI controls have been implemented in this release. A new control, the SCROLL-PANE, is a container that can host other controls. It will display scrollbars as needed to allow the user to pan the visible area.

Scroll-Pane

IsCOBOL compiler supports this new control and a new property, scroll-group, is available on child components to select their host component.

An example of a scroll-pane is shown below:

```
03 scroll-pane-1 scroll-pane
   line 8 column 2 size 68 lines 10
   transparent
   border-color rgb x#ACACAC
   .
03 scroll-pane-page scroll-group scroll-pane-1.
   05 label
      line 2 col 2 size 8 cells title "Title:"
      .
   05 entry-field
      line 2 col 12 size 52 cells id 1
   .
```

As focus changes in the child components, the isCOBOL runtime will automatically pan inside the scroll-pane to make sure the focused component is visible. Users can freely use the scroll bars and pan in the content area.

Figure 9, *Scroll-Pane control*, shows a program running with a scroll-pane container. Initially the upper left portion of the children is displayed. As the user tabs through the controls, the scroll-pane automatically scrolls to show each focused component.

Figure 10, *Scroll-Pane scrolled,* shows the scroll-pane when the last child has focus.

Figure 9. Scroll-Pane control

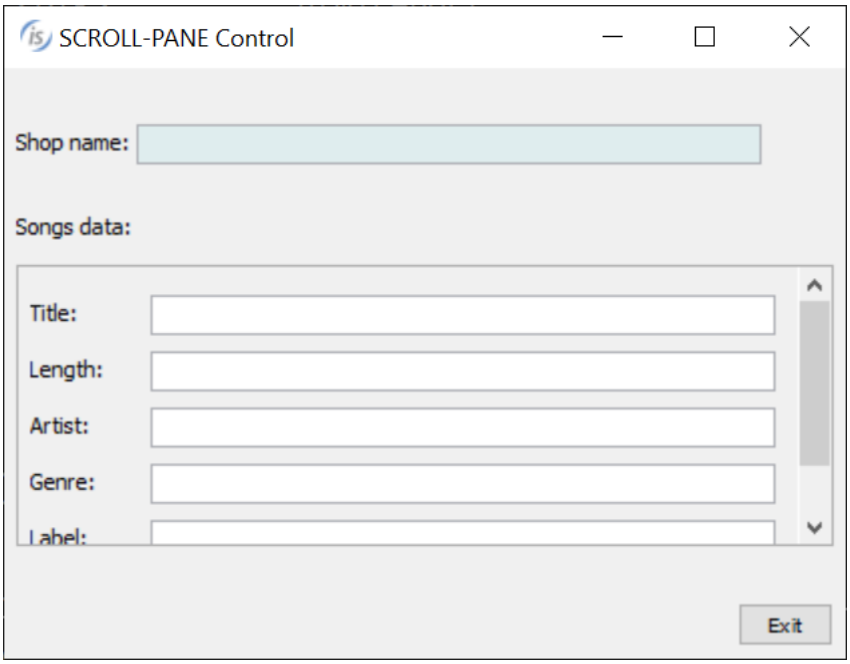


Figure 10. Scroll-Pane scrolled

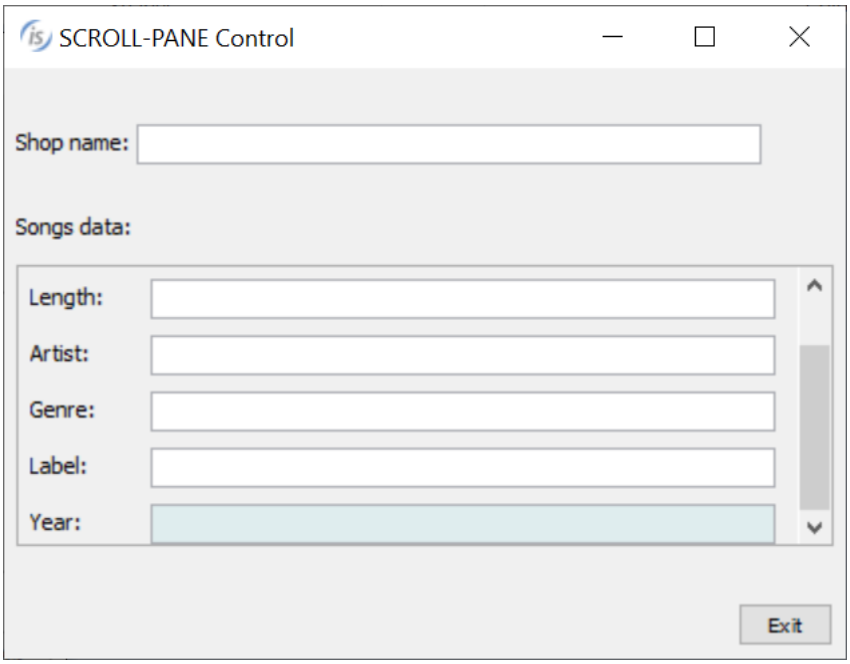


Table-View

The tree-view control now supports a new TABLE-VIEW style that activates support for column definitions inside the tree-view. This new style is useful to provide a hierarchical view with tabular data.

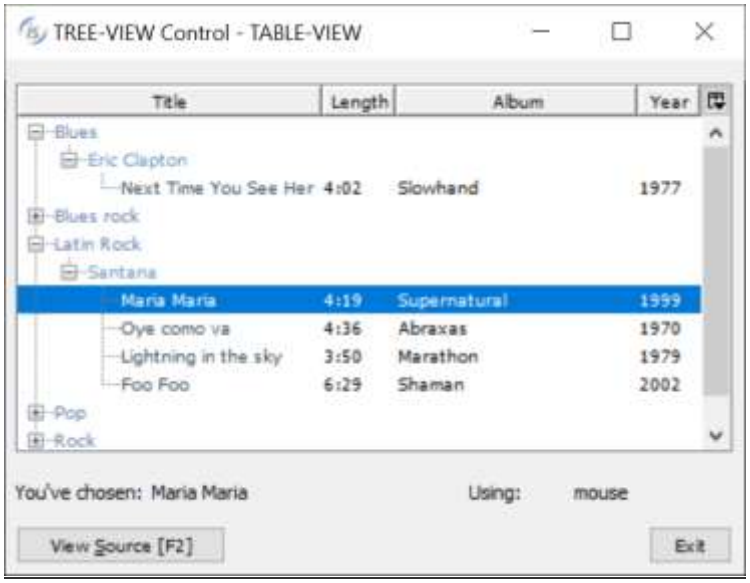
The following code snippet declares a tree-view with the table-view style:

```
01  Mask.
 03 tree-table-t tree-view table-view
     buttons lines-at-root
     line 2 col 2 lines 15 size 68 cells virtual-width 65
     display-columns (1, 30, 37, 60)
     data-columns    (record-position of rec-multi
                      record-position of rec-length
                      record-position of rec-album
                      record-position of rec-year)
     column-headings tiled-headings centered-headings
     heading-color 257 heading-menu-popup 3 end-color -16774581
     adjustable-columns reordering-columns
     event TV-EVT
 .
```

Several properties associated with column management of grid controls are supported in the table-view style, such as display-columns, data-columns, column-headings, and more.

Figure 11, *Tree-View with Table-View,* shows the new style in action.

Figure 11. Tree-View with Table-View



Other GUI enhancements

The grid control has been enhanced by adding new properties to manage grids when rows are hidden as a result of active filters, when filterable-column style is set, or because the user is using the automatic search feature, activated by pressing the keyboard shortcut Ctrl-F.

Now the visible rows can be inquired by using the property ROWS-FILTERED, as shown by the following code:

```
inquire my-grid rows-filtered in w-filtered
```

The list of returned rows is the same as with the ROWS-SELECTED property: row1 row2… rowN. If no filter is set on the grid, and there are no hidden rows, inquiring ROWS-FILTERED will return the special value -1.

The property ROWS-SELECTED has been enhanced to allow developers to set a special-value "all" in the modify statement to select all rows, as shown in the code below:

```
modify my-grid rows-selected "all"
```

The new SEARCH-PANEL property has been implemented to force the grid to show the search panel over the column headings. When set to 1, the search panel is always visible even if the grid has the NO-SEARCH style, and the user will not be able to remove it. To enable this behavior, the property can be set in the control's definition, or use the code below:

```
modify my-grid search-panel 1
```

The grid can now be configured to highlight the cell's current row and column heading with a specific color, helping the user identify the current cell content.

Three new properties can be used to configure the colors:

```
heading-cursor-background-color
```

```
heading-cursor-foreground-color
```

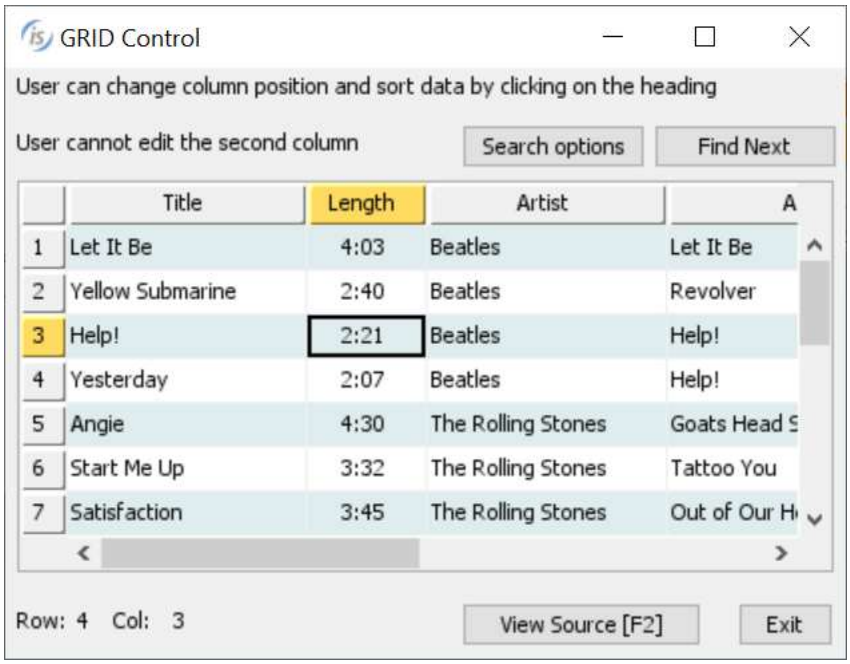
```
heading-cursor-color
```

For example, by having the following code set on the screen declaration level:

```
heading-cursor-background-color rgb x#FFDC61
```

the grid will look like popular spreadsheet programs at runtime, as shown in Figure 12, *Heading-cursor-color.* The heading colors will highlight the current focused cell position.

Figure 12. Heading-cursor-color



The new NOTIFY-MOUSE style is now supported on all controls, and is used to fire the new events MSG-MOUSE-ENTER, MSG-MOUSE-EXIT, MSG-MOUSE-CLICKED, MSG-MOUSE-DBLCLICK.

This allows developers to have finer control on the user interface and the management of mouse events, and provides more freedom in user interface design.

New GUI configurations properties available in the 2020R1 release:

- `iscobol.gui.grid.find_delay` to set the delay in milliseconds for the Grid search feature
- `iscobol.gui.messagebox.bcolor` to set the default background color of message boxes if not set explicitly on the display statement
- `iscobol.gui.messagebox.fcolor` to set the default foreground color of message boxes if not set explicitly on the display statement
- `iscobol.gui.messagebox.font` to set the default font name and size of text in the message boxes if not set explicitly on the display statement
- `iscobol.gui.windows_modality` to set the thread blocking behavior of floating windows and message boxes. This is useful on multithread applications.

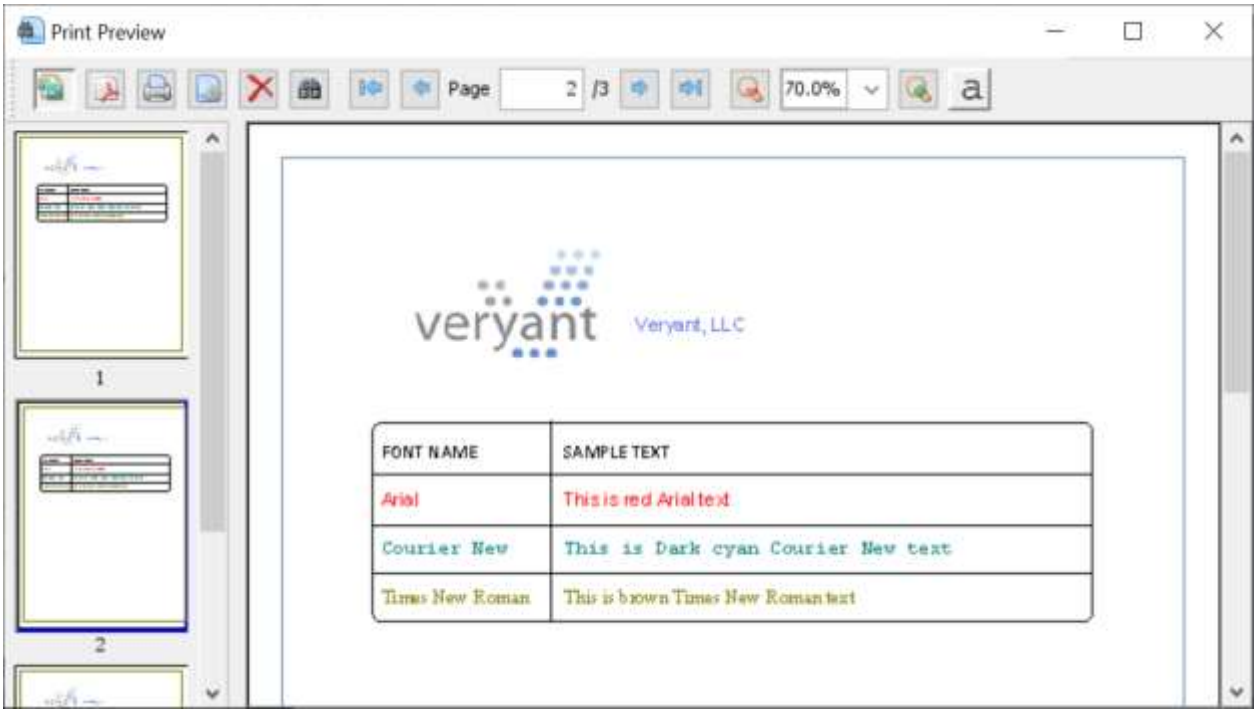
The Print Preview dialog will now show page thumbnails, enabled by default, allowing the user to quickly jump to a specific page in a multi-page report. To dynamically control the thumbnails panel, new methods have been implemented in the `com.iscobol.rts.print.SpoolPrinter` class.

```
void setShowThumbnailsButton(boolean showThumbnailsButton)
```

```
boolean isShowThumbnailsButton()
```

The new print preview with thumbnails is shown in Figure 13, *Print preview Thumbnails*.

Figure 13. Print preview Thumbnails



Database improvements

Performance of data access from Database Bridge to RDBMs tables and c-treeRTG indexed files when logging is enabled has been greatly improved. New configuration properties have been implemented in the isCOBOL runtime to better manage the "InPool" feature of c-treeRTG, which avoids the performance costs of opening an indexed file for the connected client.

Performance of Database Bridge

Performance of Database Bridge for MySQL has been improved with a new option during the EDBI generation to take advantage of MySQL hints, a feature that allows developers to control the SQL optimizer. This can be set on the edbiis wrapper level with the new –mh option or with the new compiler directive

```
iscobol.compiler.easydb.mysql.hints=true
```

A table of performance gains is shown in Figure 14, *Database Bridge for MySQL*, comparing isCOBOL 2019R2 to isCOBOL 2020R1 without and with the new –mh option.

The test was run in Windows 10 64-bit on an Intel Core i5 Processor 4440+ clocked at 3.10 GHz with 8 GB of RAM, using Oracle JDK 1.8.0_231 and MySQL 5.6. All times are in seconds. The COBOL program executes the START on different keys, then executes READ NEXT for all the records in a table containing 200,000 records.

The EDBI routine uses light cursors (`-dmld` wrapper option or compiler configuration `iscobol.compiler.easydb.light_cursors=2`), causing a new cursor to be opened every 100 records. The use of hints saves the time spent by the MySQL query optimizer to choose which index to use to scan the table.

Figure 14. Database Bridge for MySQL

operation	2019 R2	2020 R1	2020 R1 with -mh
reading records after START on Primary Key	7,51	7,29	**4,81**
reading records after START on Alternate Key	11,05	10,82	**5,53**

A new Database Bridge configuration option has been added to generate a smarter START statement that executes less query instructions:

```
iscobol.easydb.limit_dropdown=n
```

where n can be:

0=off, the default behavior

1=partial, to activate the optimization on START with the WITH SIZE clause

2=full, to activate the optimization on START without WITH SIZE clause

3=all, to activate the optimization on any START statement

As shown in Figure 15, *Database Bridge with configuration limit_drop_down*, a performance comparison is made between isCOBOL 2019R2 and isCOBOL 2020R1 without and with the new configuration. The test was run in Windows 10 64-bit on an Intel Core i5 Processor 3210M 2.50 GHz with 16 GB of RAM, using Oracle JDK 1.8.0_231 and Oracle 11 Database server. All times are in seconds. The COBOL program executes the START statement followed by a loop of READ NEXT to read the records that satisfy the conditions set on segments of the key. The table contains 100,000 records.

Figure 15. Database Bridge with configuration limit_drop_down

operation	2019 R2	2020 R1	2020 R1
configuration			iscobol.easydb.limit_dropdown=3
START on Key equal	34,04	33,06	**18,15**
START on Key not less	34,13	33,39	**18,10**

Performance of c-treeRTG indexed files

A new c-tree server configuration variable, DELAYED_DURABILITY, has been implemented in the embedded isCOBOL 2020R1 c-treeRTG version to increase performance of c-treeRTG indexed files when the logging feature is active.

Enabling logging is beneficial for a number of reasons:

- Safety: in case of file corruption caused by events such as hardware failure or unexpected power loss, when the c-treeRTG server is restarted with logging enabled, files are rebuilt automatically, ensuring data integrity, and is user-transparent.

- Replication: c-treeRTG replication requires logging to be enabled, since the replication engine relies on logging information to properly update all the servers involved. Replication can be one-directional, where data saved on a master server is replicated to a backup server, or multi-directional, where two or more servers continuously update each other in real-time. Replication has a great benefit: failover, since if a server goes offline, any other server in the replication cluster can fulfill requests for the missing server.

Since logging could slightly impact performance and increase disk usage, it is possible to enable it on only specific, highly modified files, while other less important files such as lookup tables can be used without logging to maximize performance.

The following command can be used to activate logging on a specific file:

```
ctutil -tron filename
```

All COBOL statements that update records (such as WRITE, REWRITE, DELETE) have been improved, as shown in Figure 16, *c-treeRTG logging and InPool*, a performance comparison between isCOBOL 2019R2 and isCOBOL 2020R1 without and with the new configuration variable. The test was run in Windows 10 64-bit on an Intel Core i5 Processor 3210M 2.50 GHz with 16 GB of RAM, using Oracle JDK 1.8.0_231. All times are in seconds. The indexed file contains 300,000 records.

The OPEN-READ-CLOSE test showcases the performance gains that can be obtained using the InPool feature of c-treeRTG. The test program performs a loop consisting of a single

READ inside OPEN and CLOSE statements. The configuration options that handle pooling allow developers to flexibly configure which files can use the InPool feature. These are:

```
iscobol.file.index.filepool=true

iscobol.file.index.inpool=true

iscobol.file.index.filepool.size=n

iscobol.file.index.#.inpool=true
```

Figure 16. c-treeRTG logging and InPool

isCOBOL operation	2019 R2	2020 R1	2020 R1
Ctree version:	11.6.0.64778	11.6.0.65002	11.6.0.65002
server configuration			*DELAYED_DURABILITY 1*
WRITE	97,28	96,06	51,86
READ	3,75	3,73	3,71
REWRITE	55,88	53,84	11,65
DELETE	86,09	85,75	42,93
Total	243,00	239,38	**110,15**
client configuration			*file.index.filepool=1* *file.index.inpool=1*
OPEN READ CLOSE	11,73	11,21	**1,01**

isCOBOL Compiler

Starting from isCOBOL 2020R1, the compiler supports the new POSITIONAL clause in the MOVE statement to easily move dynamic variables (X ANY LENGTH, OCCURS DYNAMIC,…) used inside structures into other structures without dynamic variables and vice versa.

This proves useful, for example, when moving data from FD declaration, which has static content, to WORKING-STORAGE items, which can contain dynamic variables, to minimize memory usage.

Intrinsic functions execution has been enhanced, and a new EFD directive has been added for temporary tables.

Move Positional

The new POSITIONAL clause is similar to the existing CORRESPONDING clause, but instead of relying on variable names to identify matching items, it will use the positional order of items to match and move data between structures. The first item in the source structure will be moved to the first item in the target structure, the second to the second, and so on.

For example, the following code snippet defines 2 structures that are comparable in the field positioning but one structure contains only fixed data items, while the other contains dynamic data items:

```
 WORKING-STORAGE SECTION.
01  struct-fix.
    03 g1-name                 pic x(50).
    03 g1-table                occurs 100.
       05 g1-account-id        pic 9(3)         value 0.
       05 g1-account-short-des pic x(20)        value space.
       05 g1-account-notes     pic x(1000)      value space.
    03 g1-address              pic x(50).
01  struct-dyn.
    03 g2-name                 pic x any length.
    03 g2-table                occurs dynamic
                               capacity cap-g2-table-occ
                               initialized.
       05 g2-account-id        pic 9(3)         value 0.
       05 g2-account-short-des pic x(20)        value space.
       05 g2-account-short-des pic x any length value space.
    03 g2-address              pic x any length.
```

The new POSITIONAL clause can be used as shown in the statement:

```
 move positional struct-fix to struct-dyn
```

and the compiler will translate that one statement in the following moves:

```
move g1-name to g2-name
 perform varying ind from 1 by 1 until ind > 100
   move g1-account-id(ind)        to g2-account-id(ind)
   move g1-account-short-des(ind) to g2-account-short-des(ind)
   move g1-account-notes(ind)     to g2-account-notes(ind)
 end-perform
 move g2-address to g2-address
```

This will make maintenance of COBOL code with large data structures much simpler.

Moreover, the DELIMITED BY DEFAULT VALUE clause can be used as in the following statement:

```
 move positional struct-fix to struct-dyn delimited by default value
```

and the compiler will translate it as:

```
move g1-name to g2-name
perform varying ind from 1 by 1 until ind > 100
  if g1-account-id(ind)        = 0 and
     g1-account-short-des(ind) = spaces and
     g1-account-short-des(ind) = spaces
     exit perform
  else
     move g1-account-id(ind)        to g2-account-id(ind)
     move g1-account-short-des(ind) to g2-account-short-des(ind)
     move g1-account-notes(ind)     to g2-account-notes(ind)
  end-if
end-perform
move g1-address to g2-address
```

This way, dynamic occurs variables will be filled until default values are found, reducing memory usage to what is actually being used.

The POSITIONAL clause can also be used when moving two fixed structures with different children and different pictures, since it relies on positional ordering to execute the moves.

Intrinsic functions

The compiler now supports a shorthand syntax to invoke functions using the $ symbol in place of the "function" keyword. This simplifies the way to execute intrinsic functions and could also be useful for compatibility with other COBOLs.

For example, the syntax:

```
display $length(g2-address)
display $capacity(g2-table)
```

is the equivalent of:

```
display function length(g2-address)
display function capacity(g2-table)
```

The two functions length and capacity will be executed during the DISPLAY statement.

Intrinsic functions can be used in any COBOL statement that requires a data item, such as MOVE, STRING, or any conditional statement, such as IF, EVALUATE, PERFORM UNITL.

EFD TEMPORARY directive

The compiler now supports a new EFD directive to mark a table as TEMPORARY instead of PERMANENT. This directive is used by the Database Bridge product. Temporary tables are useful as "working files". They are usually needed to save temporary data that is not being used concurrently and does not need to be stored permanently. Usually these temporary tables are kept in memory to provide maximum performance.

The following code snippet shows how to define the table "mywork" as temporary:

```
$efd temporary
 fd mywork.
 01 mywork-rec.
    03 mywork-k    pic 9.
    ...
```

The compiler will natively support temporary tables for any RDBMS that supports them.

Oracle 18c supports two kinds of temporary tables: global and private, and those can be specified using the `global` and `private` values in the temporary directive. Unlike traditional database servers, global temporary tables in Oracle are permanent database objects that store data on disk and objects are visible to all sessions, but data written in the tables is only visible to the session that created it. At the end of the session the data is removed, but the table remains. Private temporary tables, on the other hand, are memory-based tables, only visible to the session that created it, and are automatically dropped at the end of the session or transaction.

Example of the $efd temporary directive for Oracle 18c:

```
$efd temporary = global
```

or

```
$efd temporary = private
```

isCOBOL Debugger

Starting from isCOBOL 2020R1 the isCOBOL Debugger, used to debug ThinClient applications, makes it easier for multiple developers to debug applications running on the same Application Server. To allow this, the Application Server now spawns a separate process for each ThinClient application launched with the -d option, allowing isolation of processes so that debugging an application will not have influence on other applications that may be running in the same Application Server.

This allows debugging of applications on production servers without using a separate Application Server running on a different port number.

By default, port numbers from 9999 to 10099 are used, allowing a maximum of 100 concurrent debugging sessions. The range of ports can be configured using the new configuration variable iscobol.as.debugport_range. For example, setting the property using:

```
iscobol.as.debugport_range=20010-20013,20050
```

the Application Server will allocate ports 20010, 20011, 20012, 20013, 20050 for debugging sessions.

Using the new configuration property is transparent to the client, and there is no need to specify a port to connect to for debugging, as the server will handle it automatically. The same command line can be used to start multiple debugging sessions from multiple users, such as:

```
isclient -hostname ip-server -port 10999 -d PROGNAME
```

For compatibility, the previous command syntax forcing a specific debug port is still supported:

```
isclient -hostname ip-server -port 10999 -debugport 20012 -d PROGNAME
```

Using the isCOBOL 2020R1 release, we suggest removing the debugport option from the command line, and letting the server choose the first available debug port.

If bypassing the new Application Server behavior is needed, the following configuration variable can be set to fall back to the previous multithreaded mode:

```
iscobol.as.multitasking=0
```

isCOBOL EIS

isCOBOL EIS, Veryant's solution to write web-enabled COBOL programs, is constantly updated to provide more comprehensive web solutions. In isCOBOL 2020R1, there are many updates to WebDirect and WebClient. Also, the HTTPClient class can now consume existing web services with PUT and DELETE requests.

WebDirect enhancements

WebDirect, Veryant's technology that allows programs with screen sections to run in a web browser by converting them to HTML/CSS/Javascript equivalents, has been updated with several enhancements.

The grid control now allows keyboard navigation as its desktop counterpart, and supports inquiring the ROW-CAPACITY property.

A new configuration property has been added to allow numeric input field to display a numeric keypad on mobile devices:

```
iscobol.wd2.mobile_numpad=1
```

All layout-managers are now fully supported in WebDirect, including the "lm-responsive" layout manager. This allows developers to target multiple devices with different screen sizes and resolutions using WebDirect without having to use HTML/CSS implementation or WebClient, and to have the application respond properly when the user resizes the application or browser windows.

Figure 17, *WebDirect on Desktop*, shows the user interface of a program running on a desktop system, where each entry field is displayed next to it, since screen real estate allows it.

Figure 18 *WebDirect on smartphone*, shows the same application running on a smartphone screen, where labels are displayed above the entry fields to optimize the limited width of the screen.

Figure 17. WebDirect on Desktop

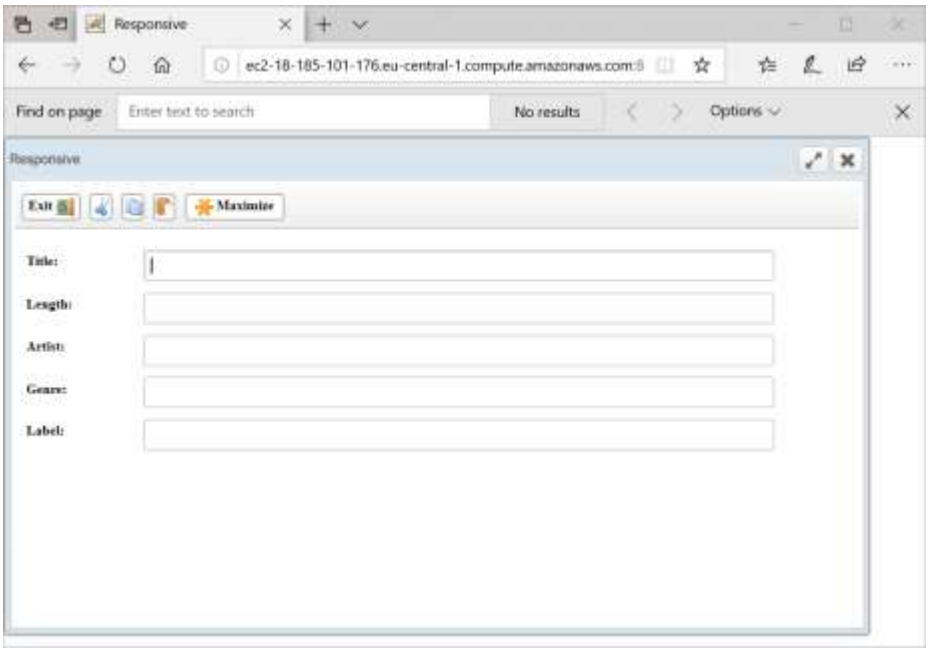


Figure 18. WebDirect on smartphone

WebClient enhancements

WebClient is Veryant's solution for running desktop applications on a remote server and interacting with it through a web browser. To interact with the application on devices that do not have a physical keyboard, WebClient provides a "Keyboard" soft button that displays the device's virtual keyboard when pressed. The keyboard now can also be activated by double tapping on the screen. A new application configuration field has been added, "*Minimum display width in pixel for keyboard button*". This allows the developer to set a minimum display resolution for the keyboard button. When the display size is less than this setting the Keyboard button is hidden, leaving all the screen real estate available to the application, and minimizing the chance of the "Keyboard" button covering the user interface elements.

Additionally, IME keyboards are now fully supported, extending support for languages such as Chinese, Japanese and Korean.

HTTPClient

HTTPClient is a class that allows COBOL programs to interact with Web Services. It has been updated to manage PUT and DELETE requests, in addition to the already implemented GET and POST requests.

The new method signatures are shown below:

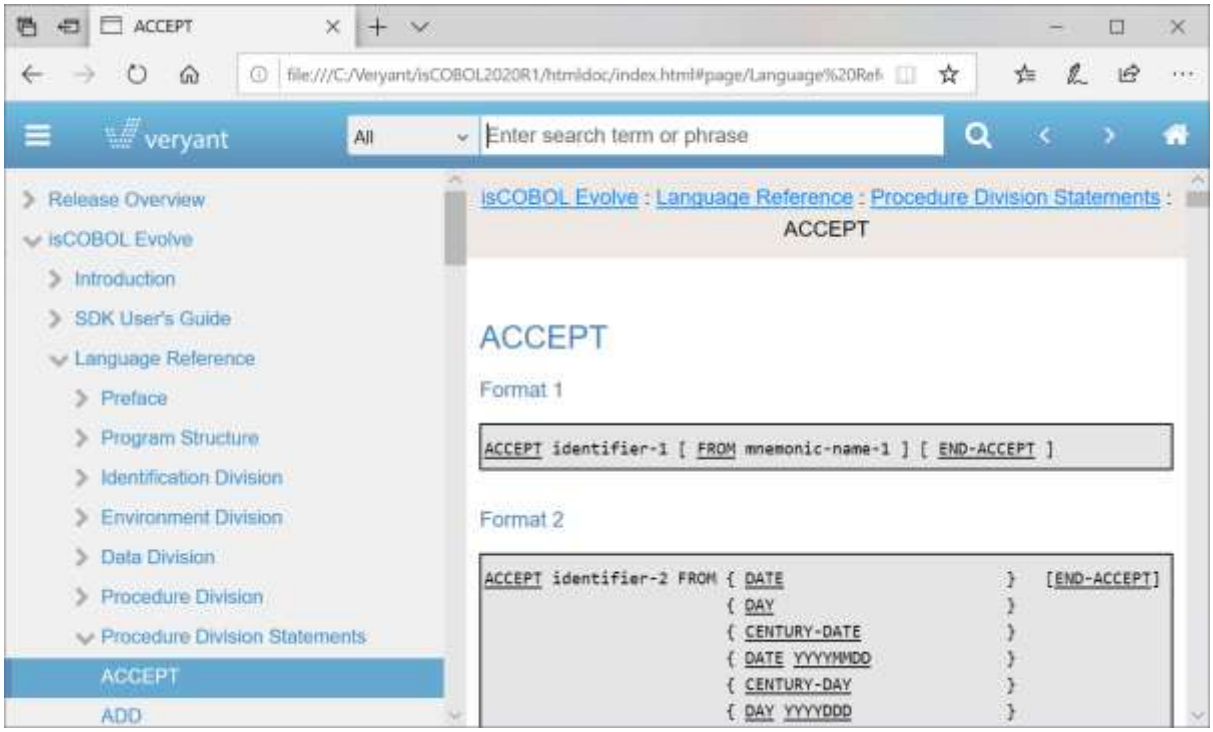
```
public void doPut(strUrl)
public void doPut(strUrl, params)
public void doPutEx(strUrl, content)
public void doPutEx(strUrl, type, content)
public void doPutEx(strUrl, type, content, hasDummyRoot)
public void doDelete(strUrl)
public void doDelete(strUrl, params)
```

Additional improvements

Since devices are becoming more and more diverse, the isCOBOL Documentation has been redesigned with a new modern look and feel and is now fully responsive, allowing it to be accessible to developers from mobile devices.

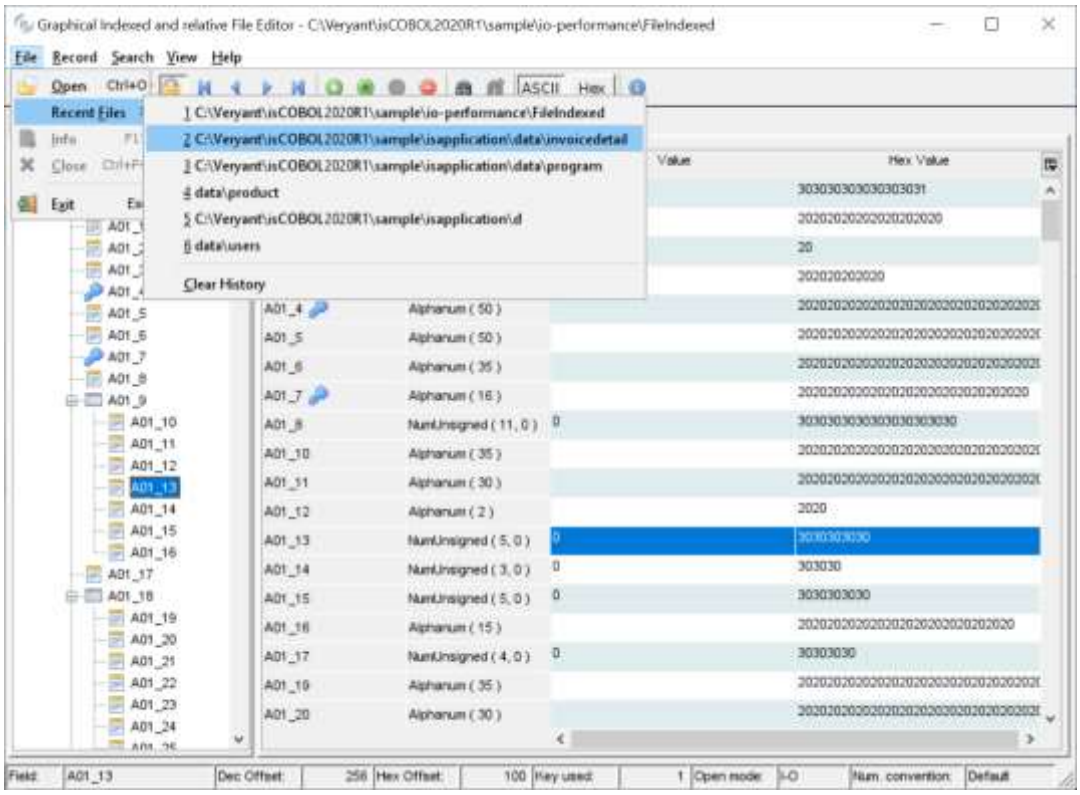
Figure 19, *New documentation*, shows the new look of the documentation.

Figure 19. New documentation



GIFE, the Graphical Indexed and relative File Editor, now manages recently opened files, making it simpler to reopen with recently used settings, such as Open Mode and Efd file. Figure 20, *GIFE recent files*, shows the new recent opened file menu items, allowing the developer to select a file with a single click. The recently used file is saved in the $user/gife.properties file when the user exits the GIFE utility.

Figure 20. GIFE recent files



The GIFE's File info dialog displays the file record size, collation and key definitions as shown in Figure 21, *GIFE File Info*.

Figure 21. GIFE File Info