isCOBOL™ Evolve

isCOBOL Evolve 2025 Release 1 Overview

isCOBOL Evolve 2025 Release 1 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL Evolve, isCOBOL Evolve 2025 R1.

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

In this version a new GUI control has been added, barcodes and QR codes can now be generated, and the search panel available in several controls has been updated.

Improvements have been developed for local variables, nested program-id and more.

isCOBOL Debugger can now display local and inline variable and supports forward and backward navigation of source code.

isCOBOL Database Bridge now supports native Java access to DBMaker using its JDBC driver.

Details on these enhancements and updates are included below.

GUI enhancements

IsCOBOL Evolve 2025 R1 introduces a new control, SPLIT-PANE, useful to enhance the visuals of GUI applications. Also, the W$BITMAP routine can now create barcodes of various types and the search panel has been updated.

SPLIT-PANE

The new SPLIT-PANE is a control that can be used to divide a screen area into two sections, either horizontally or vertically, and allows the user to resize the panes by dragging a divider. This component is useful when the User Interface requires 2 sections, for example a master-details view, and to allow the user to control how much space each section should take up.

The container can be customized using the following properties:

- DIVIDER-LOCATION to specify the location of the divider bar in percentage of the size of the Split-Pane; default: 50.
- DIVIDER-SIZE to set the size in pixels of the divider bar; default: LAF dependent.
- MIN-DIVIDER-LOCATION to set the minimum location where the user can move the divider bar; default: 0.
- MAX-DIVIDER-LOCATION to set the maximum location where the user can move the divider bar; default: 100.
- SPLIT-ORIENTATION to specify if the split is horizontal (0, default) or vertical (1).

Additional properties that should be set on the controls include:

- SPLIT-GROUP to specify the name of the split pane to which a screen section group should be added.
- SPLIT-GROUP-AREA to specify the area of the split pane that will host the group of controls: 1 is left or top, 2 is right or bottom.

With the following code snippet:

```
SCREEN SECTION.
01  Mask.
    ...
     05 split-pane-v split-pane
        line 2 column 2 size 78-split-pane-size cells lines 15.5
        divider-location     78-initial-percent-divider
        min-divider-location 20
        max-divider-location 70
        split-orientation    0
        border-color rgb     x#ACACAC
        event procedure      SP-EVENT.
        ...
     05 split-pane-v-page-1
        split-group split-pane-v split-group-area 1.
        07 ls list-box
        ...
     05 split-pane-v-page-2
        split-group split-pane-v split-group-area 2.
        07 label title "Title:"
        ...
PROCEDURE DIVISION.
...
    display Mask

SP-EVENT.
   if event-type = ntf-sp-resized
     evaluate true
     when event-data-1 > 78-initial-percent-divider
          compute list-size = 78-initial-list-size +
                  ((78-sroll-pane-size / 100) *
                   (event-data-1 - 78-initial-percent-divider))
     when ...
     end-evaluate
     display split-pane-v-page-1
  end-if.
```

a split-pane is created with the two areas divided horizontally. The left pane contains a list-box, and the right pane contains entry fields. This is the typical example of a master-details scenario, where the list contains data, and when the user clicks an item the program loads the details of the selected item and displays them in the entry-fields on the right. The user can choose the size ratio between the master-view (the list-box) and the details-view (the entry-fields).

A new event procedure, NTF-SP-RESIZED, returns the selected percentage in the event-data-1 data-item.

When dragging the bar, the dimensions of the two areas are changed. The result of the program in execution with the original size is shown in Figure 1, *The Split-Pane with original size*, while in Figure 2, *The Split-Pane after resizing*, the same screen is shown with a different split ratio for the list-box. The source code is included in the issamples folder installed with the new release.

Figure 1. The Split-Pane with original size

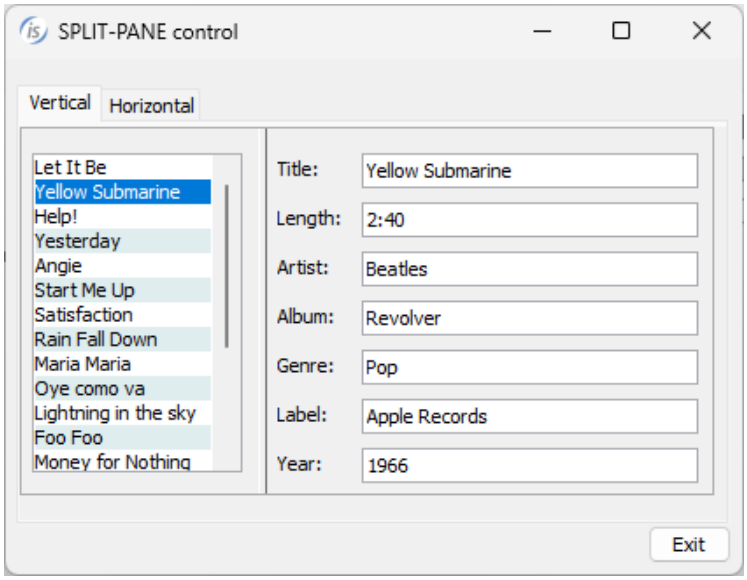
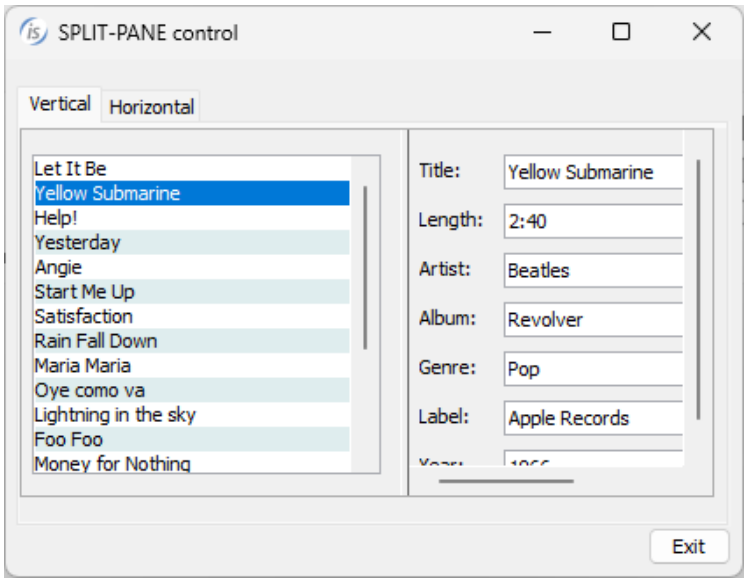


Figure 2. The Split-Pane after resizing



Barcode creation using W$BITMAP

Barcodes are very common and often need to be integrated in applications. The 2025 R1 release introduces the new WBITMAP-BARCODE-BOX in the W$BITMAP library to allow easy generation of bar codes from COBOL applications. Several barcode and output generation formats are supported.

The following is a code snippet from a sample included in the issamples folder of the new release and shows the use of the new feature:

```
call "w$bitmap" using wbitmap-barcode-box
                      p-text
                      wbitmap-bb-data
                      wrk-error-description
               giving h-bmp-barcode.
```

The generated barcode image is stored in memory to be displayed using a control that has a bitmap property or printed with the WINPRINT-PRINT-BITMAP opcode in WIN$PRINTER routine. The image can also be stored in disk file by calling the W$SAVE_IMAGE routine and can then be printed using the P$DRAWBITMAP routine or used by third party applications that require the image on disk.

Figure 3, *A QR code generation*, shows the QR code generated by passing the string www.veryant.com in the text parameter of the W$BITMAP routine.

Figure 3. A QR code generation

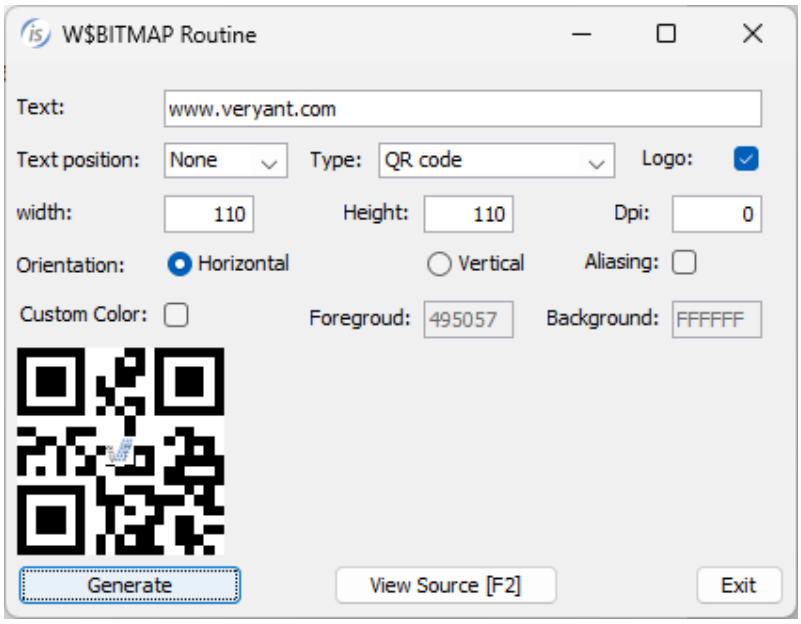
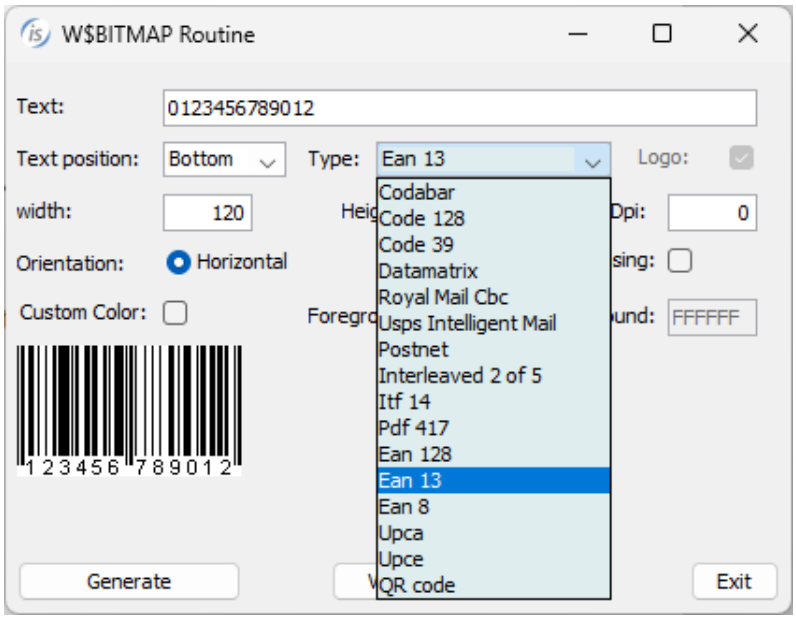


Figure 4, *EAN 13 code generation*, shows the result of the image that is generated passing a numeric value with the EAN 13 barcode type.

Figure 4. EAN 13 code generation



Customize the search panel

The search panel is an integrated component that appears by default when pressing Ctrl+F when the focus is on a control that contains multiple lines, such as grid, list-box or tree-view. The search panel can be made always visible by setting the SEARCH-PANEL to 1 on the control. On previous releases, this feature allowed users to filter the content of the control for a specific text, removing from view the lines not included in the filter. Starting from the 2025 R1 release it's possible to specify if the search text should act as a filter or just to highlight the search text in lines. It's also possible to navigate between the results with two new navigation buttons. This is like the search feature used by browsers when searching text on the page.

The new configuration option **iscobol.gui.search_panel_settings=nnnn** is used to set the visibility of items in the search panel. The value is a string of positional 0s and 1s that configuring, in order:

- Set the Filter operation mode to the filter or search behavior
- Enable Navigation buttons to change the selected search text
- Enable a Clean button to set the value of search entry to empty
- Case sensitive button to change the search criteria from sensitive to insensitive

In addition, the value 2 or 3 is supported for buttons to set the initial pressed state. The first and last buttons support these values. For example, 2 shows a visible and pushed button while 3 shows a visible but not pushed button.

The default value of this new configuration is 1012, which specifies the same filter behavior for backwards compatibility.

OOP syntax can be used to assign a different behavior to a specific control, invoking methods provided in the new com.iscobol.gui.server.SearchPanelSettings class

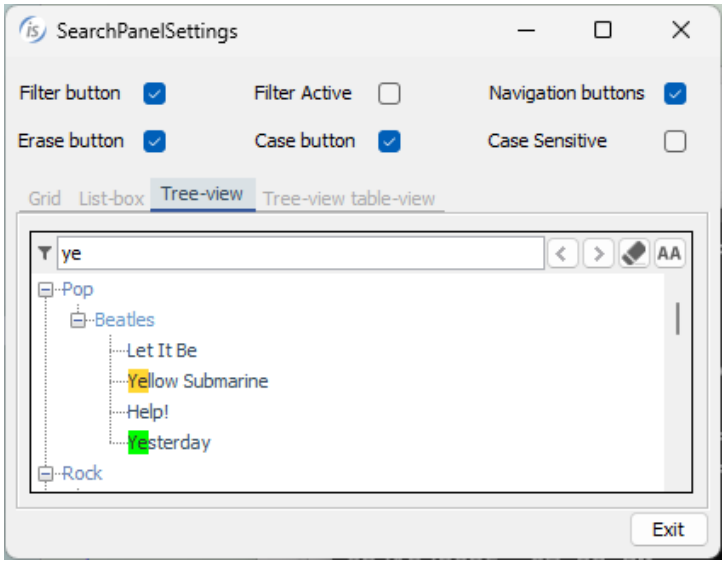
For example, the following code snippet:

```
REPOSITORY.
    class sp-settings as "com.iscobol.gui.server.SearchPanelSettings"
    class j-boolean   as "java.lang.Boolean"
    ...
SCREEN SECTION.
01  Mask.
    ...
    05 tv-songs tree-view
       ...
WORKING-STORAGE SECTION.
77  h-tv-songs            usage handle.
77  tv-songs-settings     object reference sp-settings.
PROCEDURE DIVISION.
...
    display Mask
    set h-tv-songs        to handle of tv-songs
    set tv-songs-settings to sp-settings:>new(h-tv-songs)
    tv-songs-settings:>setShowFilterButton(j-boolean:>TRUE)
    tv-songs-settings:>setFilterEnabled(j-boolean:>FALSE)
    tv-songs-settings:>setShowNavigationButtons(j-boolean:>TRUE)
    tv-songs-settings:>setShowCleanButton(j-boolean:>TRUE)
    tv-songs-settings:>setShowCaseSensitiveButton(j-boolean:>TRUE)
    tv-songs-settings:>setCaseSensitiveEnabled(j-boolean:>FALSE)
    ...
    accept Mask
    ...
```

Allows changing the visibility state of the search panel elements, using the handle for the control and calling the setters for the specific property.

Figure 5, *Customize the search panel*, shows the result of running a program with a fully customized search panel area, with the current search selection highlighted in green after pressing the navigation buttons. All the other matches are highlighted in yellow.

Figure 5. Customize the search panel

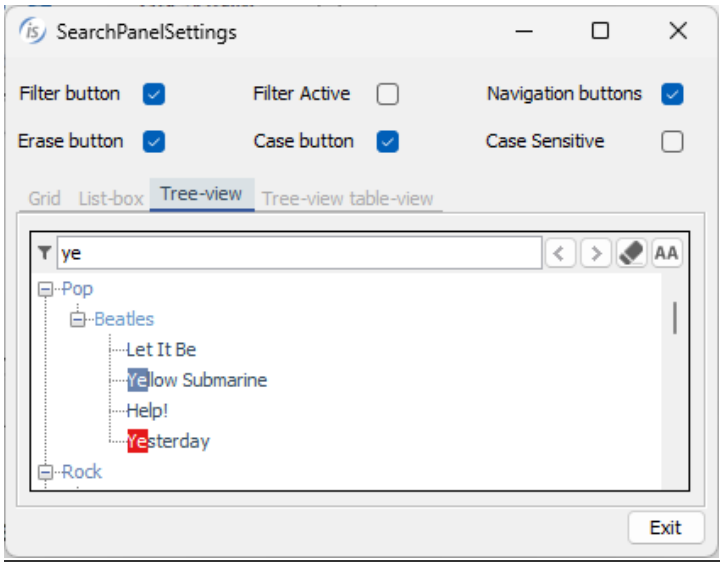


Other configurations have been implemented and improved:

`iscobol.gui.search_delay=n` (default 500) to specify when the control should start filtering data while the user interacts with the search panel in grid, list-box and tree-view controls (also known as “debounce timer”).

`iscobol.gui.matching_text_color=n;n2` to set the two colors used to highlight the searched text: the first one, before ";", is used to specify the color for all the occurrences found (default yellow), and the color after the ";", is used to color the current occurrence selected with navigation buttons (default green). For example, setting `iscobol.gui.matching_text_color= -6849454,-16777215;-15014170,-16777215` will result in using light blue and red for background and white for both foreground colors when running the program. As shown in Figure 6, *Different matching colors*, the output of the same program in execution demonstrates how to easily change colors without changing code to achieve a coherent look and feel of the entire COBOL application.

Figure 6. Different matching colors



Additional improvements

A New property, CELL-ALIGNMENT, has been implemented in the grid control to set the alignment of a specific cell. The values supported by CELL-ALIGNMENT are the same already supported in the property ALIGNMENT that is used to set on every cell in the entire column. Setting the new CELL-ALIGNMENT helps in aligning specific cells differently than the default alignment, for example to display right-aligned number cells. In the code snippet:

```
screen section.
...
   03 grid-days grid
      alignment ( "L" "R" )
      ...
procedure division.
...
    modify grid-days (3, 2) cell-alignment "C"
```

the second column has alignment set to “R”, making it right aligned, but just the column at line 3 has set alignment set to “C” which is centered.

New styles have been added in the chips-box control:

- 3-D, BOXED, NO-BOX to set different types of boxing around the control

- UNSORTED to load the chips in the order specified by the program instead of having them automatically sorted in alphabetical order.

For example, with the following code snippet:

```
screen section.
...
   03 chips-favorite chips-box
      unsorted no-box
      ...
procedure division.
...
    modify chips-favorite item-to-add "CC"
    modify chips-favorite item-to-add "BB"
    modify chips-favorite item-to-add "AA"
```

the container has the no-box style, which uses a flat look and feel, and the chips are displayed in the order in which they are inserted.

A new configuration, `iscobol.bitmap.load_method=2` allows loading legacy .BMP files that otherwise are not supported by the Java runtime. The configuration option activates a new graphics loading algorithm that can handle older formats.

The web-browser control VALUE property can now display HTML code directly instead of loading from a URL. When the value starts with "<html>" the content is assumed to be html code and not a URL.

Compiler enhancements

The Compiler supports new syntax that allows local variable declaration in a block and improves the declaration of parameters in Method-ID for Object Oriented Programming. Java classes with generic types can now be declared with a specific type, simplifying the casting using the syntax "< >". Also, concatenation using the "&" operator has been improved.

Local variable declaration

Many languages such as C# and Java support local variable declaration syntax to create a variable in a specific method or code block. This has several advantages, including:

- no risk of reusing the same variable in a different code block, creating conflicts
- thread safe code related to the variable when running multi-threading programs

The newest isCOBOL compiler implements the local variable declaration with the new DECLARE statement.

The DECLARE statement declares one or more local variables within the Procedure Division body. The scope of any inline local variable is from the point of declaration until the end of the innermost containing block. Statement clauses, paragraphs, sections and the whole method are considered to be blocks. The type of variable can be a primitive type, a java class object or any COBOL picture declared under a TYPEDEF clause. A code snippet like:

```
repository.
    class jstring as "java.lang.String"
    ...
working-storage section.
01  type-count  pic 9(9) comp typedef.
...
procedure division.
...
    if var1 = 1
       declare temp as jstring
       move "ABC" to temp
       ...
    else
       declare temp as type-count = 0
       move 123 to temp
       ...
    end-if.
```

shows the temp variable declared in two different blocks; the first inside the IF where the variable is declared as java.lang.String, the other inside the ELSE where the variable is declared as “pic 9(9) comp” that is the picture used in the assigned TYPEDEF.

Shorter declaration of parameters in Method-ID

The Compiler supports a new syntax to define method parameters directly in the METHOD-ID paragraph, without using the Linkage Section and the USING clause after PROCEDURE DIVISION. This is typically useful in class-id with many methods that receive and return parameters. It’s now possible to specify the method signature directly in the METHOD-ID. A code like:

```
method-id. setAccount as "setAccount" (par1 as JString,
                                       par2 as JString,
                                       par3 as JInt).
procedure division.
```

is equivalent to this:

```
method-id. setAccount as "setAccount".
linkage section.
77 par1 object reference JString.
77 par2 object reference JString.
77 par3 object reference JInt.
procedure division using par1,
                         par2,
                         par3.
```

The advantage is shorter code and a source that looks more similar to other languages, for example in Java the equivalent code is:

```
public static void setAccount(java.lang.String par1,
                              java.lang.String par2,
                              java.lang.Integer par3)
```

Improved Generic type parameter support

Java generics are a powerful feature that allows you to write more flexible and reusable code. They enable you to define classes, interfaces, and methods with type parameters, which can be specified when you instantiate or invoke them.

Generics help maintain consistency in your code by ensuring that the same type is used throughout a collection or method. This reduces the likelihood of errors and makes the code easier to understand and maintain.

Here are some key benefits of using generics:

- Enhanced Readability: By specifying the type of elements a collection can hold, generics make the code more readable. It becomes clear what type of objects are expected, which helps other developers (or your future self) understand the code more quickly.
- Type Safety: Generics ensure that you catch type errors at compile time rather than at runtime. This reduces the risk of ClassCastException and makes your code more robust.
- Code Reusability: With generics, you can write a single class or method that works with different types. This eliminates the need for multiple versions of the same code.
- Elimination of Casts: Generics allow you to avoid explicit casting, making your code cleaner and easier to read.
- Improved Performance: Since generics provide compile-time type checking, they can help optimize performance by reducing the need for runtime type checks.

Here's a simple example to illustrate the use of Java generics:

```
configuration section.
repository.
    class ArrayListS as "java.util.ArrayList<java.lang.String>"
    class ArrayListI as "java.util.ArrayList<java.lang.Integer>"
...
working-storage section.
77 stringList  object reference ArrayListS.
77 integerList object reference ArrayListI.
...
procedure division.
    ...
    set stringList to ArrayListS:>new()
    stringList:>add("AA")
    stringList:>add("BB")
    stringList:>add("CC")
    set integerList to ArrayListI:>new()
    integerList:>add(1)
    integerList:>add(2)
    integerList:>add(3)
```

In this example, we create two ArrayList instances: one for String objects and another for Integer objects. The use of generics ensures that only the specified type of elements can be added to each list, providing type safety and eliminating the need for explicit casting.

The compiler uses the "< >" syntax in the class name to specify the type of the “generics” arguments.

Concatenation using “&”

The Compiler already supports the syntax “&” to concatenate two or more string literals. Starting from the 2025 R1 release concatenation is also supported between data-items, method results and resource strings, making it much more useful and versatile. It allows you to reduce the code that typically requires additional variable definitions and the STRING statement.

For example, the following code snippet:

```
repository.
    class jstring as "java.lang.String"
    ...
working-storage section.
77  var    pic x any length.
77  dest   pic x any length.
77  objstr object reference jstring.
    ...
procedure division.
...
    move "String literal" & var
                          & r"myres"
                          & objstr:>toUpperCase()
                       to dest
```

shows that a string literal is concatenated with a data item, a resource that is declared in the file loaded with the configurations iscobol.resource.file, iscobol.resource.country and iscobol.resource.language and the values returned by the call to the toUpperCase method.

Compatibility improvements

isCOBOL 2025 R1 has been enhanced to improve compatibility with other COBOL dialects such as MicroFocus COBOL and IBM COBOL. The syntax improvements in the compiler are related to nested programs and XML. A new library routine has been added.

Nested programs

Multiple programs can be included in the same source file repeating the PROGRAM-ID / END PROGRAM syntax. There are two different scenarios:

- If a program is included before the END PROGRAM clause of another program, it becomes a nested program, and it can be called only by its parent program.
- If a program is included after the END PROGRAM clause of another program, it becomes a sibling program, and it can be called by every other program in the runtime session.

For example, in the code:

```
PROGRAM-ID. customer.
WORKING-STORAGE SECTION.
...
PROCEDURE DIVISION.
    ...
    EXIT program.
END PROGRAM customer.

PROGRAM-ID. product.
WORKING-STORAGE SECTION.
...
PROCEDURE DIVISION.
    ...
    CALL "show" USING ...
    ...
  EXIT program.

  PROGRAM-ID. show.
  WORKING-STORAGE SECTION.
  ...
  LINKAGE SECTION.
  ...
  PROCEDURE DIVISION USING ...
  ...
  END PROGRAM show.

END PROGRAM product.
```

There are three declared PROGRAM-IDs; “customer” and “product” that are sibling programs and can be called as normal programs, and “show”, which is a program-id declared inside the “product” program and therefore can only be called inside the “product” program. Executing the same CALL to “show” in a different program causes the runtime error “CALL not found” to be raised.

XML improvements

XML PARSE is a statement used to parse an XML document into its individual components that are then passed, one at a time, to a user-written processing procedure. The XML GENERATE statement converts data to XML format. In version 2025 R1 the additional clauses VALIDATING and ENCODING are now supported to improve compatibility and enhance the feature:

- VALIDATING will validate the XML file during the PARSE using a schema file declared in the SPECIAL-NAMES
- ENCODING will set the encoding to be used during the XML PARSE and XML GENERATE statements

For example, the following code snippet:

```
CONFIGURATION SECTION.
SPECIAL-NAMES.
     XML-SCHEMA XMLSCHART IS 'XMLSCHART.XSD'.
PROCEDURE DIVISION.
...
  XML GENERATE OUTPUT-XML FROM INPUT-XML
       COUNT IN XML-SIZE
       WITH ENCODING 1208
  END-XML
...
  XML PARSE OUTPUT-XML
       VALIDATING WITH FILE XMLSCHART
       WITH ENCODING 1208
       PROCESSING PROCEDURE EVENT-HANDLER
  END-XML
```

declares the XML-SCHEMA named XMLSCHART under SPECIAL-NAMES, assigning it to a physical schema (.xsd) file. During the XML PARSE, the XMLSCHART schema is passed to the VALIDATING clause. Both statements, XML GENERATE and XML PARSE use the ENCODING clause to force a different encoding than the one used when running. In this case 1208 means UTF-8.

Library routine

A new library routine named CBL_LOCATE_FILE has been implemented to enhance compatibility with MicroFocus COBOL. The CBL_LOCATE_FILE routine has two uses: it can be used to expand an environment variable in a file specification, where the environment variable contains a list of several paths. It can also determine whether an OPEN INPUT statement using a particular file specification finds the file on disk.

A code snippet like this:

```
WORKING-STORAGE SECTION.
77  user-file-spec   pic x(128).
77  user-mode        pic x comp-x.
01  actual-file-spec.
    03 buffer-len    pic x(2) comp-x.
    03 buffer        pic x(128).
77  exist-flag       pic x comp-x.
77  path-flag        pic x comp-x.
77  status-code      pic xx comp-5.
...
PROCEDURE DIVISION.
...
    move "$PATH\dyncall.dll" to user-file-spec
    move 0 to user-mode
    initialize actual-file-spec
    move 128 to buffer-len
    call "CBL_LOCATE_FILE" using user-file-spec
                                 user-mode
                                 actual-file-spec
                                 exist-flag
                                 path-flag
                       returning status-code
```

checks if the dyncall.dll library is in the PATH.

ESQL enhancements

The new 2025 R1 release improves compatibility with ESQL Precompilers, such as ProCobol and DB2prep. The embedded SQL language has been improved by adding support of SAVEPOINTs and LOB locator.

SAVEPOINT support

A savepoint in SQL is a mechanism that allows you to set a specific point within a transaction to which you can later roll back. This feature is particularly useful for implementing partial rollbacks in case of errors or other exceptional conditions within a transaction. The below procedural code demonstrates the feature:

```
*working in autocommit-off mode
*insert one record
   EXEC SQL
      INSERT INTO TEST_TABLE (COL_1) VALUES ('aaa')
   END-EXEC
*define a savepoint
   EXEC SQL
      SAVEPOINT SP1
   END-EXEC.
*insert another record
   EXEC SQL
      INSERT INTO TEST_TABLE (COL_1) VALUES ('bbb')
   END-EXEC
*rollback to savepoint
   EXEC SQL
      ROLLBACK TO SAVEPOINT SP1
   END-EXEC.
*check table content
   EXEC SQL
      DECLARE CUR SCROLL CURSOR FOR
         SELECT COL_1 FROM TEST_TABLE
   END-EXEC
   EXEC SQL
      OPEN CUR
   END-EXEC
   PERFORM UNTIL SQLCODE = 100
      EXEC SQL
         FETCH NEXT CUR INTO :WRK-COL1
      END-EXEC
      DISPLAY WRK-COL1
   END-PERFORM
   ...
```

When running this code, the program will display only 'aaa' because the insertion of 'bbb' has been cancelled by the ROLLBACK statement, as the ROLLBACK statement will cancel all operations that occurred after the specified SAVEPOINT.

SQL TYPE improvement

LOB (Large Objects) can now be mapped to host variables through locator variables with an IBM DB2 compliant syntax. BLOB, CLOB, and DBCLOB are data types used in databases to store large amounts of data. BLOB is used for binary data like images, audio, and video, making it ideal for multimedia files. CLOB is designed for large text documents, storing character data such as XML or JSON. DBCLOB is tailored for double-byte character data, which is common in languages like Chinese, Japanese, and Korean, making it suitable for large text data in these languages. Each type is optimized for different data formats to ensure efficient storage and retrieval.

The newly added SQL syntaxes are:

```
dataitem SQL TYPE IS BLOB-LOCATOR.
dataitem SQL TYPE IS CLOB-LOCATOR.
dataitem SQL TYPE IS DBCLOB-LOCATOR.
```

The example below demonstrates how to read the content of a CLOB column using a CLOB-LOCATOR on IBM DB2:

```
WORKING-STORAGE SECTION.
01  LOB-LOCATOR USAGE SQL TYPE IS CLOB-LOCATOR.
01  LOB-BUFFER  PIC X(128).
...
PROCEDURE DIVISION.
...
    EXEC SQL
         DECLARE C1 CURSOR FOR
            SELECT CLOB_COLUMN INTO :LOB-LOCATOR
                   FROM LOB_TABLE WHERE LOB_ID = 1
    END-EXEC
    EXEC SQL
         OPEN C1
    END-EXEC
    EXEC SQL
         FETCH C1 INTO :LOB-LOCATOR
    END-EXEC
    EXEC SQL
         SELECT :LOB-LOCATOR
         INTO :LOB-BUFFER
         FROM SYSIBM.SYSDUMMY1
    END-EXEC
```

Although this is DB2 compliant syntax, it can also work on other databases if they support the three kinds of large objects associated with the locator. For example, the above code can work also in Oracle after changing the SYSIBM.SYSDUMMY1 to DUAL.

Runtime enhancements

The isCOBOL runtime has been enhanced with a new option to measure the time spent, new configuration options and others.

-time option

A new option, named –time, is supported by the iscrun process to measure the time spent by the running session. This is useful in batch programs whose runtime needs to be measured. The same option is supported by the trun process.

For example, running the command:

```
iscrun -time IO_INDEXED
```

The output is:

```
INDEXED FILES
NUM-TIMES:  10000
...
Total time elapsed: 3.60 seconds
```

The last line shows the total time elapsed, including the time spent for Java startup and shutdown.

New configurations:

A new property, `iscobol.runtime-preload`, has been implemented to preload a list of programs separated by space or comma. This feature is similar to the C$PRELOAD routine that loads all the COBOL programs contained in a jar library or in a folder, but you can use this configuration to list arbitrary program names without the need to create a separate jar file or a folder to group the programs.

The property also helps in cases where a PROGRAM-ID contains ENTRY points, and the caller program calls the entry directly without ever calling the main program. This approach is the equivalent of C functions called in a native library and preloaded with the configuration iscobol.shared_library_list.

For example, having these entry points declared in this PROGRAM-ID:

```
program-id. progentry.
procedure division.
main.
...
entry "entry1".
...
entry "entry2".
...
```

a program can now execute:

```
  call "entry1"
  call "entry2"
```

without adding the CALL “progentry” before calling the entry points by setting the configuration property:

```
iscobol.runtime.preload=progentry ...
```

A new property `iscobol.file.env_toupper=false` has been implemented to resolve file aliases in case sensitive way. When searching for an environment variable, its name is considered in upper case by default. By setting this new configuration to false the runtime will look for the environment variable in case sensitive way, keeping the case used in the program source code. It affects only environment variables in file names, for example with this select:

```
select customers assign to "$datapath/customers"
```

the runtime will search for the file “customer” in the $datapath environment variable instead of $DATAPATH. The same behavior is applied to file names remapped in the environment when having `iscobol.file.env_naming=true`.

Other improvements

The WIN$PLAYSOUND routine now supports files included in the source with COPY RESOURCE statement. This avoids the need to provide the physical file in the production environment since the resource file is included in the .class when compiling the COBOL source.

The USE AT PROGRAM START declaratives have been enhanced by supporting the linkage parameters and library routines. The statement is useful to provide startup code that will be executed only once when the program is instantiated.

New methods are implemented in the com.iscobol.rts.IscobolSystem class. The method signatures are:

```
public static void duplicateIscobolEnv(Thread src, Thread dst)
public static void destroyIscobolEnv(Thread th)
```

The duplicateIscobolEnv method is useful to duplicate the isCOBOL environment between Java threads. The destroyIscobolEnv method is used to clear the isCOBOL environment of a Java thread. These methods help in the COBOL integration with Java for multithread environments that require an isolated environment, similar in the COBOL solution under the CALL RUN statement.

Debugger enhancements

The isCOBOL Debugger has been improved, adding support for inline and local variables, improved code navigation and other minor improvements.

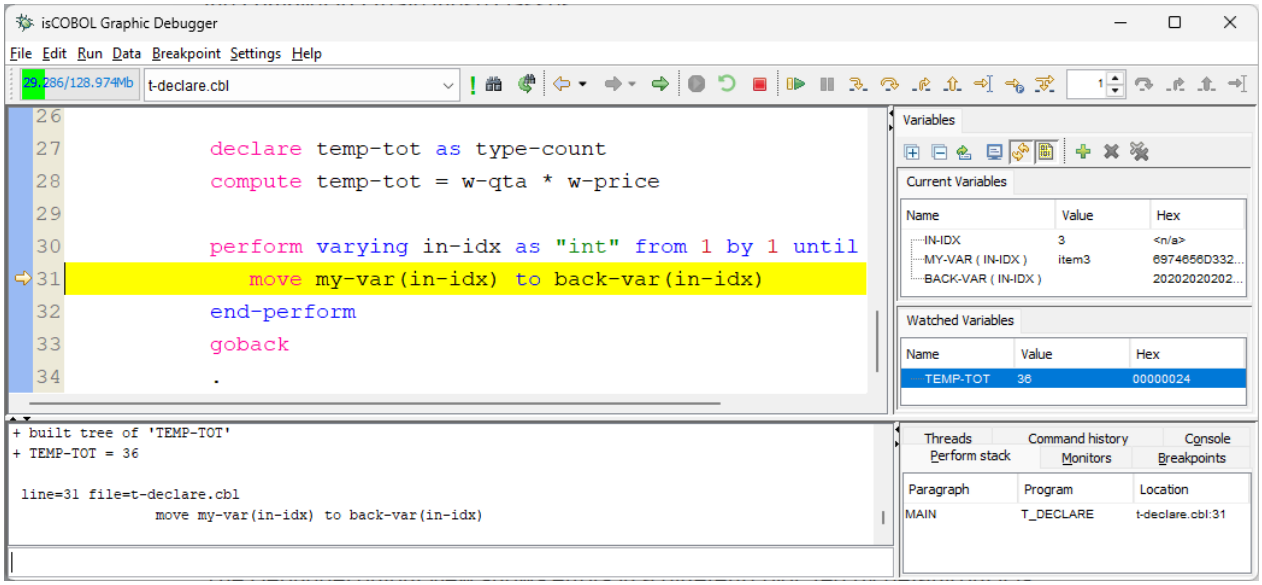
Inline and local variables

Inline declared items and local variables using the DECLARE statement are now fully supported in the debugger. When compiling a source that contains DECLARE statements or inline items in debug mode, the compiler generates additional .class files that start with the same program name and have a suffix like $LocalVars$n.

The debugger uses these classes to access local variables. Source code needs to be compiled using the -d or -dx options (debug compilation).

As shown in Figure 7, *Access to local variables from Debugger*, the local variables are handled as regular variables, and they are integrated in Current Variables, in the Watched variables, etc... helping developers in the debugging activity.

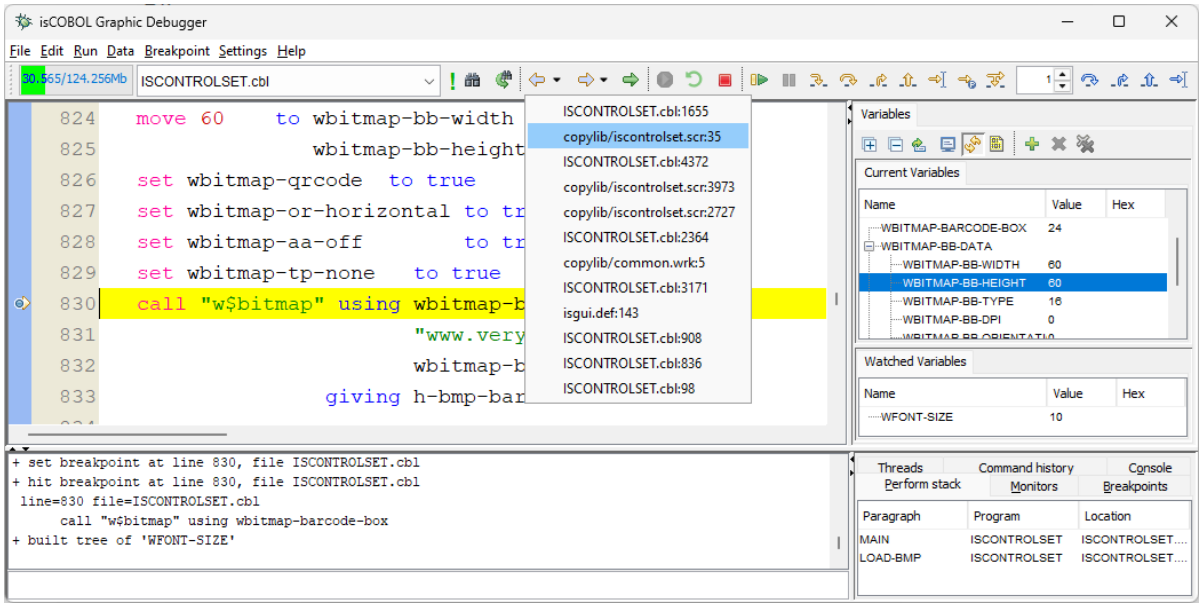
Figure 7. Access to local variables from Debugger



Other improvements

Two new buttons have been added in the Debugger tool-bar, and are used to execute the Back and Forward commands in the history of selected lines with the feature “Go to declaration”. Navigating to declaration can also be performed by double clicking on a paragraph name or a data item or using the hyperlink declaration. This feature simplifies source code navigation, allowing you to re-select a previously selected line, similar to how the isCOBOL IDE implements it in the Eclipse environment. As shown in Figure 8, *Back history feature*, the history of the navigated lines is listed in a pop-up menu, and selecting an item from the list will load the corresponding source file at the specified line.

Figure 8. Back history feature



The new RESTART command can be used to stop and automatically restart the debugging session. The command is also available in the Run menu and in a new tool-bar button.

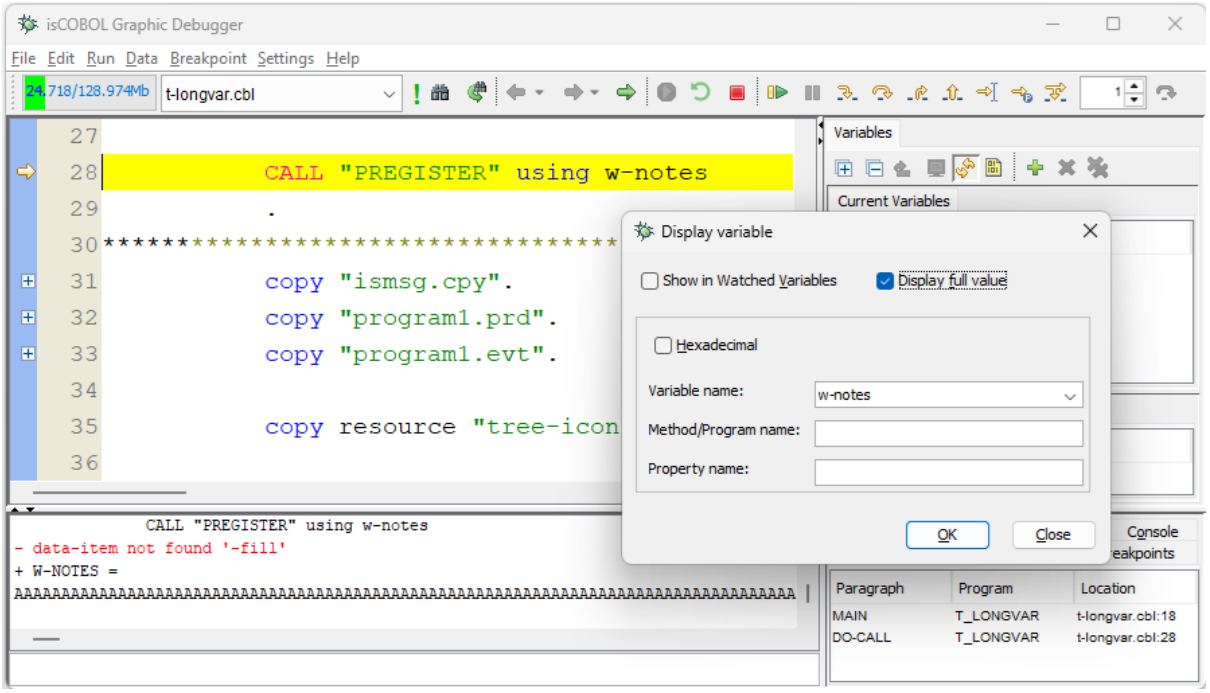
By default, the Output view shows the first 1024 characters of a data item, but the existing Display and Env commands have been improved with the addition of the -full option, which will display the whole content of large data items.

In addition, the graphical window “Display variable” now contains a new check-box to specify the –full option.

The Debugger Output view now shows errors using a different color, red by default but it can be customized in the “Fonts and Colors” settings.

As shown in Figure 9, *-full option*, the new option is used to show the content of the large data item and the previous command executed with a typo error, -fill instead of –full, is shown in the Output view in red text.

Figure 9. –full option



IsCOBOL Database Bridge

isCOBOL Database Bridge is the tool that enables COBOL programs to interact with RDBMs without changing COBOL source code or learning ESQL. It has been enhanced to support a new relational database: DBMaker.

DBMaker is a powerful and flexible SQL Database Management System (RDBMS) that supports an interactive Structured Query Language (SQL), a Microsoft Open Database Connectivity (ODBC) compatible interface, and Embedded SQL for C (ESQL/C). DBMaker also supports a Java Database Connectivity compliant interface and DBMaker COBOL interface (DCI). The unique open architecture and native ODBC interface give you the freedom to build custom applications using a wide variety of programming tools or to query databases using existing ODBC-compliant applications. DBMaker is easily scalable from personal single-user databases to distributed enterprise-wide databases. The advanced security, integrity, and reliability features of DBMaker ensure the safety of critical data. Extensive cross-platform support permits you to leverage existing hardware, allows for expansion and upgrades to more powerful hardware as your needs grow. DBMaker provides excellent multimedia handling capabilities to store, search, retrieve, and manipulate all types of multimedia data. Binary Large Objects (BLOBs) ensure the integrity of multimedia data by taking full advantage of the advanced security and crash recovery mechanisms included in DBMaker. File Objects (FOs) manage multimedia data while maintaining the capability to edit individual files in the source application.

isCOBOL already supported the DCI interface to access DBMaker's tables using statements like OPEN, READ, WRITE, using two configurations:

`iscobol.file.index=dci` to use the DCI C client library in the same Java process. This is typically used in standalone processes

and

`iscobol.file.index=dcic` to use the DCI C client library in a separated process that is called DCI connector. This is typically used in multithread environments like isCOBOL Server with multiple ThinClient or WebClient connections.

The isCOBOL Database Bridge now supports the DBMaker's JDBC driver, making it a better choice when deploying in environments such as Tomcat or Java Servlet containers, where a 100% Java access is preferable.

To generate EDBI routines for DBMaker, use the following Compiler configuration:

```
iscobol.compiler.easydb=1
```

```
iscobol.compiler.easydb.dbmaker=1
```

Routines for DBMaker have the dbm prefix in their name; for example, having a file whose physical name (or EFD FILE directive value) is "articles", the Compiler will generate a routine named dbmEDBI-articles.cbl.

Routines for DBMaker can be also generated with the legacy approach, through the edbiis command, that is now improved with a new option: **-dd**.

By processing EFD dictionaries with the new -dd option, you obtain EDBI routines without the dbm file suffix; for example, given the articles.xml EFD dictionary file, the edbiis command will generate a routine named EDBI-articles.dbm.

Although EDBI routines for DBMaker can be used with both the legacy type 3 JDBC driver (dmjdbc30.jar) and the new type 4 JDBC driver (dmjdbct4.jar), it is suggested to use EDBI routines for DBMaker with the new type 4 JDBC driver, as it is a pure Java driver without any native dependency, and therefore is suitable for platforms where there's no porting of the DBMaker client (i.e. MacOS) and environments where native components are not always welcomed, like Java Servlet containers.

At runtime, all you need to do is activate the EasyDB file handler as usual by adding the configuration:

```
iscobol.file.index=easydb
```

```
iscobol.file.prefix=dbm
```

If your programs operate on platforms or environments where native components are supported you might consider continuing use of the DCI file handler instead.