isCOBOL™ Evolve

isCOBOL Evolve 2026 Release 2 Overview

isCOBOL Evolve 2026 Release 2 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL Evolve, isCOBOL Evolve 2026 R2.

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

The 2026R2 release improves GUI components by supporting a new control named PROGRESS-BAR and new isCOBOL beans to provide additional GUI components.

Enhancements are also available in the OOP syntax and support for parameterized sections.

Several improvements have been implemented in Visual Studio Code extension and isCOBOL utilities.

Details on these enhancements and updates are included below.

GUI enhancements

isCOBOL Evolve 2026 R2 supports a new control named PROGRESS-BAR and new isCOBOL beans to expand the set of GUI components. Additional improvements have been implemented in existing controls to support modern components like TOGGLE buttons.

PROGRESS-BAR control

A progress bar is a visual UI component that displays the completion status of an ongoing task, such as software installation and long running processes. It improves user experience by managing waiting expectations, reducing uncertainty, and assuring users that the system is actively processing. The new control named PROGRESS-BAR can be declared in the screen section or created with a DISPLAY statement; by default, it is created horizontally and can be set vertically by setting the style to VERTICAL.

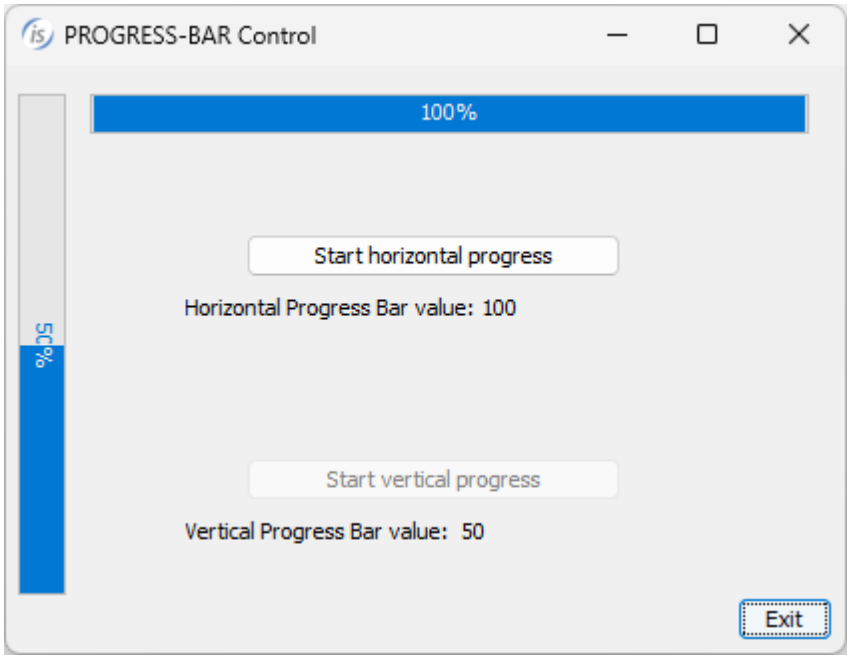
The following code snippet:

```
01  Mask.
    03 v-pr-bar progress-bar vertical
       line 2 col 2 size 4 lines 18.
    03 h-pr-bar progress-bar
       line 2 col 8 size 60 lines 1.5.
...
    perform varying perc-h from 10 by 10 until perc-h > 100
       ...
       modify h-pr-bar value perc-h
    end-perform
```

declares two progress-bar components in a screen. In a processing loop in the procedure division, the value of the control is updated with a MODIFY statement to update the progress status.

The output is shown in Figure 1, *Progress-bar control*.

Figure 1. Progress-bar control.



TOGGLE button

A toggle button is a graphical control element that allows users to switch between two exclusive states, such as On/Off. They have been made popular on mobile platforms like Android and iOS but are now considered standards for modern and recent graphical applications on both Web and Desktop. Since the logic of a toggle is similar to a check-box, a new style named TOGGLE has been implemented in the check-box control. By setting the style to TOGGLE the component will be rendered accordingly. The new style also allows you to easily convert all check-box controls used in an entire COBOL application in toggle buttons by using the compiler code injection feature.

In addition to customizing the look of the components, an additional property named TOGGLE-ATTRIBUTES has been implemented to specify several attributes using a syntax that resembles CSS styles: a list of names-value pairs can be specified separating each with a semicolon.

The list of current supported attributes is:

- **disabled-thumb-background-color-off** to set the color of the disabled thumb in OFF state
- **disabled-thumb-background-color-on** to set the color of the disabled thumb in ON state
- **disabled-track-background-color-off** to set the background color of disabled toggle track in OFF state
- **disabled-track-background-color-on** to set the background color of disabled toggle track in ON state
- **on-icon** to set the symbol shown inside the thumb when the status is ON
- **thumb-background-color-off** to set the color of the thumb in OFF state
- **thumb-background-color-on** to set the color of the thumb in ON state
- **thumb-radius** to set the radius of the thumb
- **thumb-transition** to set the animation time taken to switch state on click
- **track-background-color-off** to set the background color of the toggle track in OFF state
- **track-background-color-on** to set the background color of the toggle track in ON state
- **track-radius** to set the radius of the track border

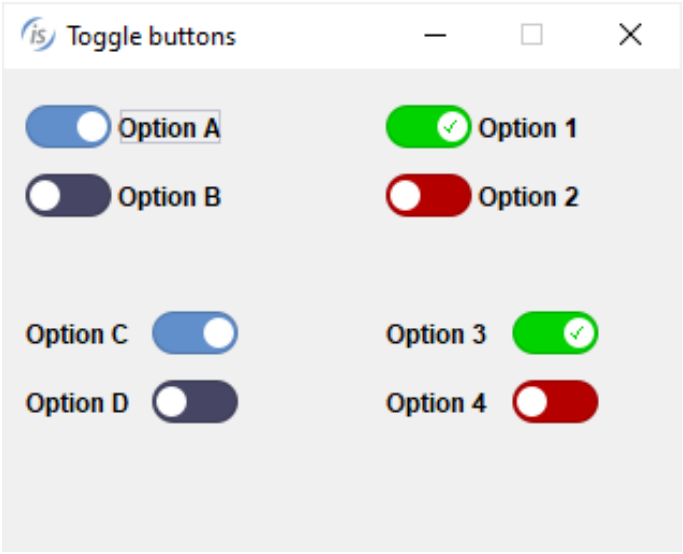
The following code snippet:

```
01  Mask.
     03 Cb-toggle-A check-box toggle
        line 2 col 2 size 15 cells title "Option A" value var-ta.
     03 Cb-toggle-B check-box toggle
        line 4 col 2 size 15 cells title "Option B" value var-tb.
     03 Cb-toggle-C check-box toggle
        line 8 col 2 size 15 cells title "Option C" value var-tc
        left-text left-text-alignment 1.
     03 Cb-toggle-D check-box toggle
        line 10 col 2 size 15 cells title "Option D" value var-td
        left-text left-text-alignment 1.
     03 Cb-toggle-1 check-box toggle toggle-attributes toggle-attrs
        line 2 col 26 size 15 cells title "Option 1" value var-t1.
```

```
       03 Cb-toggle-2 check-box toggle toggle-attributes toggle-attrs
          line 4 col 26 size 15 cells title "Option 2" value var-t2.
       03 Cb-toggle-3 check-box toggle toggle-attributes toggle-attrs
          line 8 col 26 size 15 cells title "Option 3" value var-t3
          left-text left-text-alignment 1.
       03 Cb-toggle-4 check-box toggle toggle-attributes toggle-attrs
          line 10 col 26 size 15 cells title "Option 4" value var-t4
          left-text left-text-alignment 1.
...
 move "track-background-color-on: #00D300; " &
      "thumb-background-color-on: #FFFFFF; " &
      "track-background-color-off: #B50000; " &
      "thumb-background-color-off: #FFFFFF; " &
      "on-icon: '\2713'; thumb-transition: 0.3s; " &
      "thumb-radius: 100%; track-radius: 100%" to toggle-attrs.
```

declares several check-box controls with the new TOGGLE style in a screen, the ones placed on the left section use the default colors, while the ones placed on the right section use the new property named TOGGLE-ATTRIBUTES to customize colors, and display an icon in the On state. The output is shown in Figure 2, *Toggle buttons*.

Figure 2. Toggle buttons.



IsCOBOL Beans

Several new controls have been implemented and are provided as standard java-beans components. They are compatible with both ThinClient and WebClient environments.

The new components are:

- DropZone to allow drag-and-drop of files in a zone

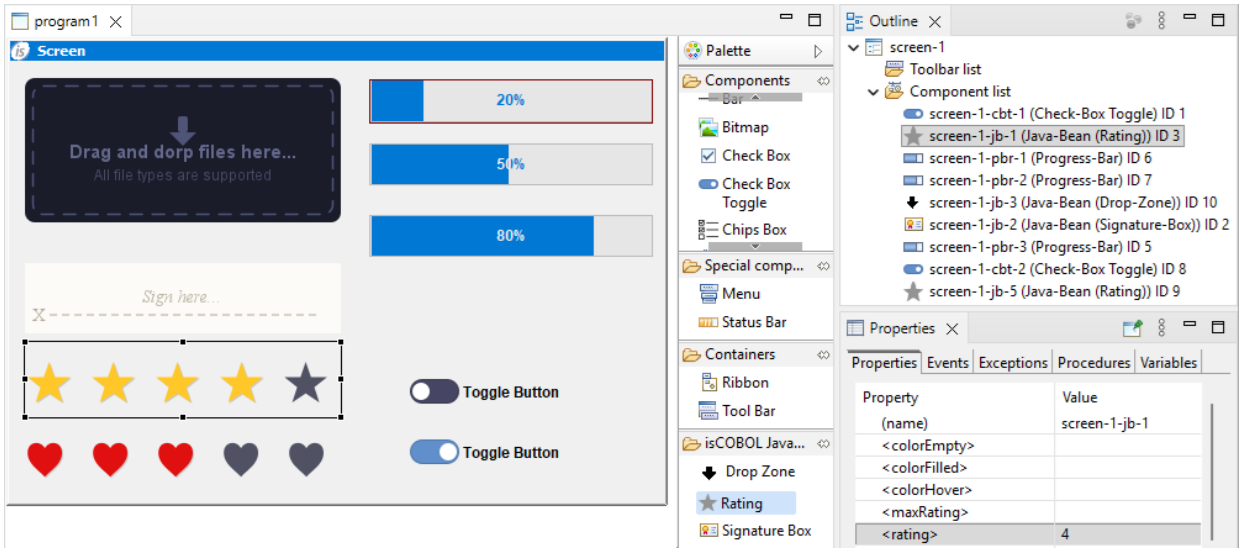
- Rating for requesting feedback using a simple visual scale, to make it easy for users to communicate their assessment with minimal effort.

- SignatureBox to draw a signature in an area

A sample of each component can be found in the issamples folder.

The new java-bean controls are integrated into the IDE Screen Designer to simplify their management for COBOL developers as shown in Figure 3, *isCOBOL Beans in IDE Painter*.

Figure 3. isCOBOL Beans in IDE Painter.



DropZone

The DropZone component is an interface element that allows users to easily drag and drop files into an application. It simplifies interactions by providing a designated visual area where users can drag files from their desktop or click to open the Open-Save-Box dialog to explore and choose the content of folders. It is typically used to upload files in a Web application, but it can be used also on Desktop application to interact with files.

Specific properties can be set to customize colors and text that appear in the box.

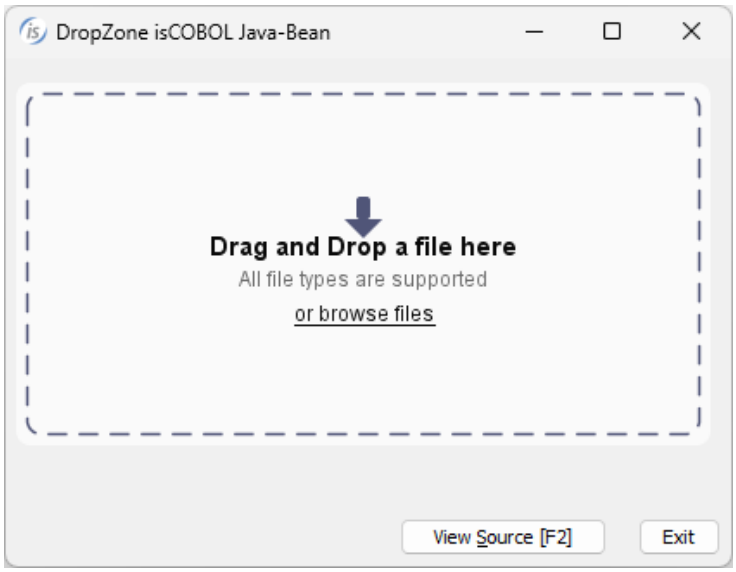
When a file is dragged to the component, an event occurs and can be intercepted from the program to retrieve the filename, allowing the program to control what action needs to be executed as a response. A code like the following:

```
REPOSITORY.
    class JavaBean as "com.iscobol.gui.server.CobolGUIJavaBean"
    ...
WORKING-STORAGE SECTION.
77  DropZone                   object reference JavaBean.
...
SCREEN SECTION.
...
    03 Jdropzone java-bean
       clsid "com.iscobol.misc.javabeans.DropZone"
       object DropZone
       line 2 col 2 lines 15 size 67
       event-list ("propertychange") event DROPZONE-EVT.
PROCEDURE DIVISION.
...
    DropZone:>setProperty("backgroundColor",
              j-color:>new(250 as int, 250 as int, 250 as int))
    DropZone:>setProperty("backgroundHoverColor",
              j-color:>new(223 as int, 237 as int, 238 as int))
    DropZone:>setProperty("textColor",
              j-color:>new(10 as int, 10 as int, 10 as int))
    DropZone:>setProperty("textHoverColor",
              j-color:>new(100 as int, 160 as int, 255 as int))
...
DROPZONE-EVT.
    if event-type = msg-jb-event
       if event-data-2 = iscobol-jb-propertychange
          set pc-object to e-object as prop-change-evt
          if pc-object:>getPropertyName() = "droppedFiles"
             set file-path to DropZone:>callMethod("getDroppedFiles")
             display message "Selected file:" file-path
          end-if
       end-if
    end-if.
```

declares the necessary classes in REPOSITORY and the necessary object reference in WORKING-STORAGE. The java-bean control is then declared In the SCREEN SECTION, where the clsid property is set to the class of the isCOBOL bean. In the PROCEDURE code, setProperty methods are executed to customize colors and text, and in the event procedure the list of dropped files is retrieved and shown in a message box.

The output is shown in Figure 4, *DropZone bean*.

Figure 4. DropZone bean.



Rating

The rating component is a user interface element that allows users to provide feedback or express satisfaction by assigning a score, often on a fixed scale like stars, hearts, or any other symbol. The visual changes that occur upon hovering or clicking provide immediate feedback on a user's selection, then the underlying numerical value (typically from 1 to 5) can be retrieved and processed as needed.

This bean has specific properties that can be used to customize the colors and symbols used in the rating. The rate set by the user can be retrieved from code or in an event handler code that is fired when the user clicks on the component.

A code like this:

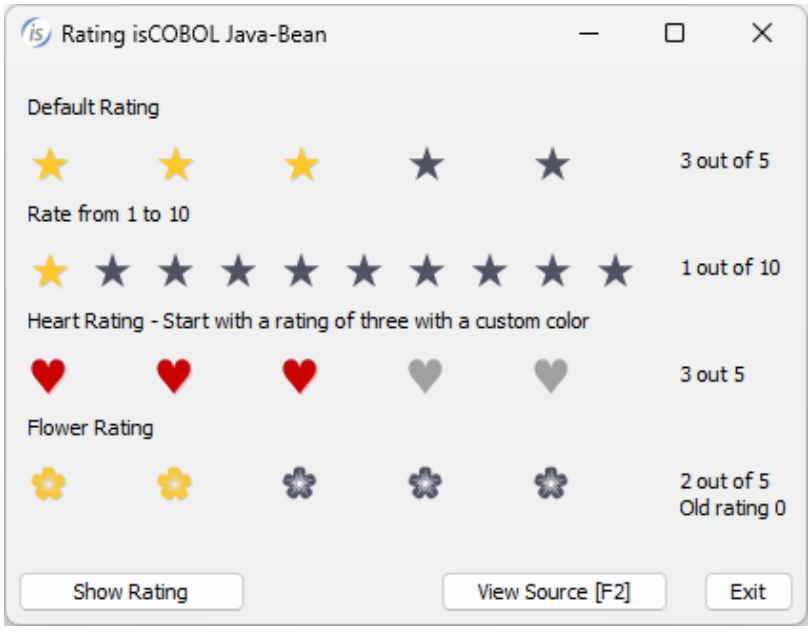
```
SCREEN SECTION.
...
  03 star-rating java-bean clsid "com.iscobol.misc.javabeans.Rating"
     line 7 col 3 size 55 object StarRating
     ...
  03 heart-rating java-bean clsid "com.iscobol.misc.javabeans.Rating"
     line 11 col 3 size 55 object HeartRating
     ...
PROCEDURE DIVISION.
...
  StarRating:>callMethod("setMaxRating", 10 as int).
  StarRating:>callMethod("setRating", 2 as int).

  HeartRating:>callMethod("setSymbol", nx"2665").
  HeartRating:>callMethod("setRating", 3 as int).
```

declares two Rating components in the SCREEN SECTION, and in the PROCEDURE code, 'callMethod' methods are executed to customize the maximum rating and current rating in both the star-rating and the symbol in the heart-rating.

The output of the program, which is installed in the issamples folder, is shown in Figure 5, *Rating bean*.

Figure 5. Rating bean.



SignatureBox

The SignatureBox component is a user interface element that allows users to digitally draw or type their signature directly into an application. It is commonly used for applications that handle documents or data where collecting a signature is required. Users can sign using a mouse on desktops or via touchscreens and styluses on mobile devices, then the component converts the drawing into a digital format, usually saving it as an image file.

Specific properties can be set to customize colors and text that appear in the box, to save the image on demand and to clear the signature in the box.

Code such as:

```
SCREEN SECTION.
...
  03 Jsignbox java-bean clsid "com.iscobol.misc.javabeans.SignatureBox"
     line 2 col 3 lines 15 size 66 object SignBox
     ...
  03 pb-save push-button
     line 20 col 2 size 15 cells title "Save signature"
     exception-value 102 enabled e-save self-act
  03 pb-reset push-button
     line 20 col 19 size 15 cells title "Reset signature"
     exception-value 104 enabled e-save self-act.
     ...
PROCEDURE DIVISION.
...
INIT.
    SignBox:>setProperty("strokeColor"
                 j-color:>new(0 as int, 0 as int, 0 as int))
    SignBox:>setProperty("baseStroke", 2 as float)
    SignBox:>setProperty("backgroundColor",
             j-color:>new(250 as int, 250 as int, 250 as int))
    SignBox:>setProperty("baselineColor",
                 j-color:>new(10 as int, 10 as int, 10 as int))
    ...
RESET-SIGNATURE.
    SignBox:>callMethod("clear")
    ...
SAVE-SIGNATURE.
    set buffimg to SignBox:>getProperty("signatureImage") as j-buffimg
    try
        buffimg:>writeImage("png", img-to-save as string)
    catch exception
        display message exception-object
    end-try
    ...
```

creates a SignatureBox component in SCREEN SECTION, then in the PROCEDURE code, properties are set to customize colors. The user can press buttons to reset and save the signature as needed.

The output of the program is shown in Figure 6, *SignatureBox bean*.

Figure 6. SignatureBox bean.

New components in grid cells

The new progress-bar control and new bean components are also available to use in grid cells using the DISPLAY UPON syntax. This allows for simple placement of the new components in all cells of a specific column or in a specific cell. For example, with the following code:

```
display check-box toggle  upon g1(-1, 1)
display progress-bar      upon g1(-1, 3)
display java-bean  clsid "com.iscobol.misc.javabeans.Rating"
                         editor-on-click 1 upon g1(-1, 4)
...
perform varying ind ...
  inquire g1(ind, 4) cell-handle jb-handle
  inquire jb-handle object obj-rating
  invoke obj-rating:>callMethod("setRating", w-rat(ind - 1))
end-perform
```

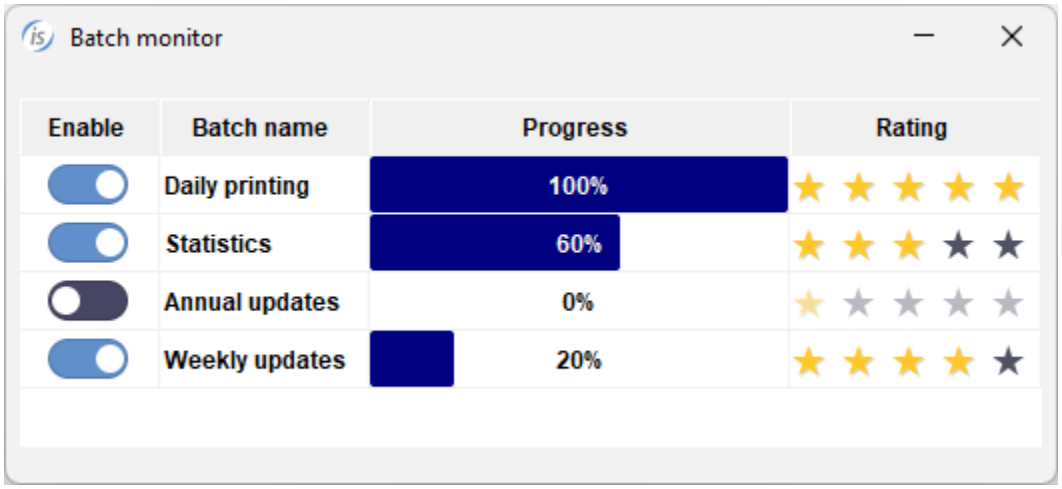
a Toggle component is created in the first column, a Progress-Bar is created in the third column, and a Rating bean is created in the last column. The property EDITOR-ON-CLICK

can be set during the control creation to customize the behavior of the editor control that will require only one click to be activated.

A program can loop and read the new CELL-HANDLE property to retrieve the handle of the component and inquire or set data as necessary.

The output of the code is shown in Figure 7, *New components in the grid*.

Figure 7. New components in the grid.



In addition, new properties are implemented to set the cell color during the mouseover action. The names of the properties are CELL-ROLLOVER-COLOR, CELL-ROLLOVER-BACKGROUND-COLOR and CELL-ROLLOVER-FOREGROUND-COLOR. They work like the existing ROW-ROLLOVER-COLOR but they set the color of a single cell instead of the entire row. The new properties can be set directly upon control creation using the DISPLAY statement or can be changed during runtime execution using the MODIFY statement. The following code snippet shows a grid declared in SCREEN SECTION with the rollover color effect on the cell level:

```
03 gd1 grid
       ...
       cell-rollover-background-color rgb x#B7DFC9
       cell-rollover-foreground-color rgb x#217346
       .
```

W$BITMAP new op-code

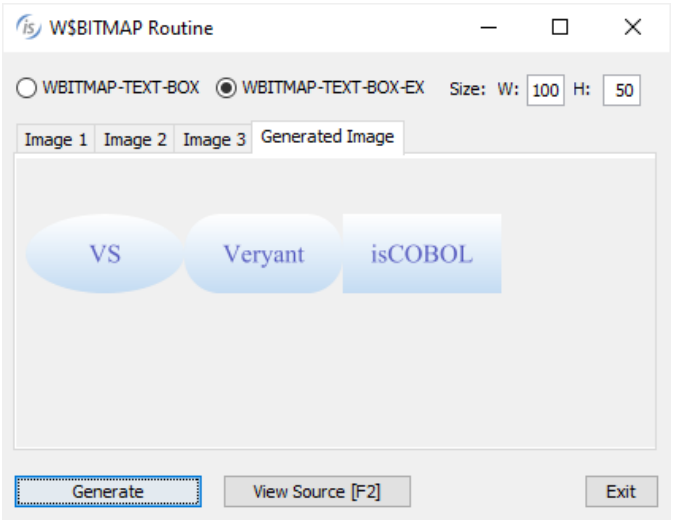
W$BITMAP's WBITMAP-TEXT-BOX opcode, available since the 2024 R2 release, allows developers to generate squared or circular icons. In this release, a new op-code, named WBITMAP-TEXT-BOX-EX, is supported to generate rectangles and rounded or oval icons in a strip of images.

For example, with the following code:

```
initialize wbitmap-tb-data-ex
move 100        to wbitmap-tb-width-ex
move  50        to wbitmap-tb-height-ex
move "VS"       to wbitmap-tb-text-ex(1)
move "Veryant"  to wbitmap-tb-text-ex(2)
move "isCOBOL"  to wbitmap-tb-text-ex(3)
set wbitmap-tb-rect-ex(2) to true
set wbitmap-tb-rect-ex(3) to true
move 90         to wbitmap-tb-radius-ex(2)
move h-font     to wbitmap-tb-font-ex(1), ...
move wrk-color  to wbitmap-tb-text-color-ex(1), ...
move gradient-north-to-south to wbitmap-tb-grd-or-ex(1), ...
move wrk-color2 to wbitmap-tb-bg-color-2-ex(1), ...
call "w$bitmap" using wbitmap-text-box-ex, wbitmap-tb-data-ex
                giving h-image-icon
```

The structure wbitmap-tb-data-ex is set to create a strip of bitmap with 3 images of different types. The output of the code is shown in Figure 8, *New WBITMAP-TEXT-BOX-EX op-code.*

Figure 8. New WBITMAP-TEXT-BOX-EX op-code.



Other graphical improvements

New properties are now supported in date-entry control to customize colors:

- WEEKDAY-FOREGROUND to set the color of weekdays
- SUNDAY-FOREGROUND to set the color of Sunday
- VALID-DATE-FOREGROUND to set the color of valid dates
- INVALID-DATE-FOREGROUND to set the color of invalid dates

This snippet shows the declaration of a date-entry control with the new properties in SCREEN SECTION:

```
03 de-contr Date-Entry numeric
   line 5 col 32 size 20
   weekday-foreground     11
   sunday-foreground      12
   valid-date-foreground  14
   invalid-date-foreground 15
```

A new style, named ISLINK is available for push-button controls to render the button as a hyperlink.

A new property named ISLINK-ATTRIBUTES has been implemented to customize the layout of push-button controls with the ISLINK style.

The property value consists of a sequence of attribute-value pairs expressed in a CSS-like syntax. Attributes and their values are separated by a colon (:), while individual attribute-value pairs are separated by a semicolon (;).

The currently supported attributes are:

- **disabled-hover-background-color** to set the background color of a disabled button on mouse over
- **disabled-hover-color** to set the text color of disabled button on mouse over
- **disabled-link-background-color** to set the background color of disabled button on mouse out
- **disabled-link-color** to set the text color of disabled button on mouse out

- **hover-background-color** to set the background color on mouse over
- **hover-color** to set the text color on mouse over
- **hover-decoration** to set the text decoration when the mouse is over the control. Possible values are 'underline' or 'none'
- **hover-font-weight** to set the text weight on mouse hover. Possible values are 'normal' or 'bold'
- **link-background-color** to set the background color on mouse out
- **link-color** to set the text color on mouse out
- **link-decoration** to set the text decoration on mouse out. Possible values are 'underline' or 'none'
- **link-font-weight** to set the text weight on mouse out. Possible values are 'normal' or 'bold'

The following code snippet declares some buttons controls with the new ISLINK style and new ISLINK-ATTRIBUTES property in a screen, then the content of the variable is assembled in the procedure code to set the link effects of color, the underline decoration and bold font on hover action:

```
03 push-button islink islink-attributes pb-attrs
   line 20 col  2 size 10 title "Load" exception-value 3.
03 push-button islink islink-attributes pb-attrs
   line 20 col 12 size 10 title "Save" exception-value 4.
03 push-button islink islink-attributes pb-attrs
   line 20 col 32 size 10 title "Exit" exception-value 27.
...
 move "hover-color: #0000FF; " &
      "hover-decoration: underline; " &
      "hover-font-weight: bold" to pb-attrs
```

The output of the code is shown in Figure 9, *ISLINK in action*, where the mouse is over the Save button so all the characteristics described in the attributes are applied on the button.

Figure 9. ISLINK in action.

The LAST-ITEM property, already available in the chips-box control, is now also supported on combo-box and list-box controls, and is used to inquire the number of items in a control, as shown in the following snippet:

```
inquire cb1 last-item w-combo-items
inquire lb1 last-item w-list-items
```

New configuration options are also supported in this release:

iscobol.gui.mouse_shape_on_buttons=true/false to configure the option to change the shape of the mouse to a hand when hovering over button controls - push-button, check-box (including the ones with TOGGLE style) and radio-button. By default, it is set to be true.

iscobol.whint.tolerance=n to specify how many pixels the mouse must move to make a hint disappear. The default value is 20.

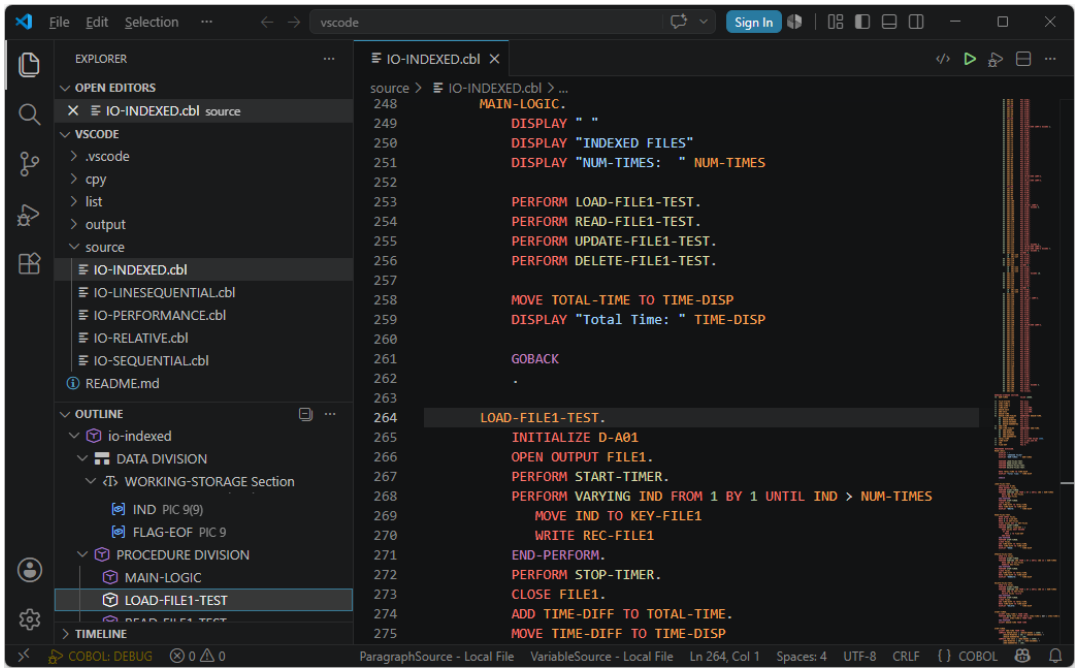
Visual Studio Code Extension

The IsCOBOL 2026 R2 release contains an updated version of the Microsoft Visual Studio Code extension that includes new features, such as support for Outline view and Formatter, better integration of COPY files in IntelliSense features such as variable and paragraph completion. The newest version of the extension adds support for external "Pre-processors" that can be configured in a new section of the Settings page.

Outline view

The Outline View in Visual Studio Code is a navigation panel that displays the hierarchical structure of the currently active file. It lets you quickly understand and navigate the symbols defined in the program without having to scroll through the source code. The isCOBOL 2026 R2 release of the extension includes a new Outline View that parses the currently active source file and its copybook files, and displays the structure of the code, allowing the user to click an element of the outline view and jump to the corresponding source code line, as shown in Figure 10, *Outline view*.

Figure 10. Outline view.



The Outline view can help a developer get a clearer picture of the file and help in navigating complex code.

Code Formatter

The Format Document feature in Visual Studio Code automatically reformats the contents of a file or the selected code according to language-specific formatting rules. Its purpose is to improve readability and maintain a consistent coding style without changing the program's behavior.

COBOL is one of the most challenging languages to format automatically because its syntax, historical source formats, and coding conventions are very different from modern block-structured languages.

The new Visual Studio Code extension in the 2026 R2 release implements source code indentation for isCOBOL source code.

Document formatting provides several advantages:

- improves readability and maintainability
- ensures consistent coding style across a team
- reduces time spent on manual formatting
- minimizes formatting-related differences in version control
- allow developers to focus on code rather than layout

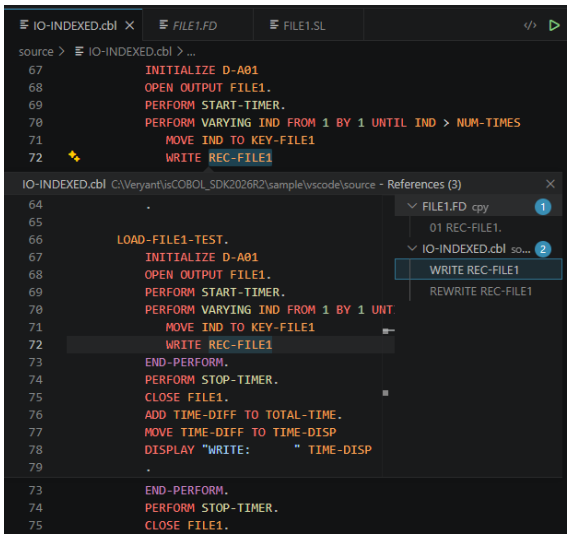
IntelliSense with COPY files

IntelliSense is Visual Studio Code's collection of language-aware editing features that help developers write code faster, more accurately, and with fewer errors. Rather than being a single feature, IntelliSense encompasses code completion, hints, hover information, symbol navigation, code actions, and other language-specific capabilities. The 2026 R2 release of the extension expands the support for items declared in COPY files and supports the COPY REPLACING, COPY... OF and COPY SUPPRESS syntaxes. Data items or paragraphs declared in copybook files and in COPY codebooks are now parsed and available when using the code completion features of the editor.

Also new in this release is the support for variables and paragraphs defined in COPY files for the “Goto definition”, “Find references” and “Peek” commands. If a variable is defined or used in a COPY file, the name of the copy and its containing folder are displayed.

Figure 11, *Peek references*, shows the result of right-clicking on a variable and selecting the context menu Peek / Peek references popup. Clicking on the variable names in the popup allows you to quickly jump in the relevant source file.

Figure 11. Peek references.



Additional settings

The new “Environment Variables” setting is now available in Visual Studio Code’s settings and can be used to set environment variables before launching a COBOL application from the editor. The Copylib-folders setting lets you specify a list of folders where copybook files are stored. Both relative and absolute paths to folders can be added to the list.

Code pre-processors are now supported and can be configured in the “Veryant / Preprocessor: command” setting. You specify the command line for the pre-processor that receives the COBOL source file as first parameter and the output file as second parameter.

When compiling, each source file is handed to the preprocessor for processing, and the resulting output file is compiled.

isCOBOL Compiler

isCOBOL Evolve 2026 R2 supports the parameterize sections that are useful to perform sections passing parameters and returning a result. The new compiler version also supports the ternary operator and provides improvements in the OOP syntax.

Parameterized sections

Parameterized sections in modern COBOLs refer to a language extension that allows program sections to accept arguments and return values. They behave like reusable functions or methods but reside directly within the PROCEDURE DIVISION of legacy PROGRAM-ID source code. The input arguments are defined directly in the section header in the parenthesis. The number of parameters and their type define the method's signature. The output return type and value are defined directly in the header in the RETURNING clause. Input and output types must be declared as typedef or class name defined in the repository, much like you do in the method-id signature using the short syntax.

A parametrized section is executed like a call to an intrinsic function, for example in a MOVE statement. If the parametrized section does not return a value, it can be called using a PERFORM statement, passing the list of arguments within parenthesis.

The main advantages that parameterized sections offer are:

- avoiding the need to create a separate subprogram
- keeping variables declared as arguments and return values are scoped locally, enabling recursion and thread-safe code
- producing clear compiler errors when a section is executed without passing the proper arguments
- allowing legacy systems to be exposed and maintained as modern, API-like interfaces

In modern COBOL implementations, a recursive section for a Fibonacci sequence looks like this:

```
program-id. Fibonacci.
working-storage section.
77 binary-long is typedef signed-long.
procedure division.
    display fib-sect(10)
    goback.
fib-sect section (n as binary-long) returning res as binary-long.
    if n <= 1
       move n to res
       exit section
    end-if
    compute res = fib-sect(n - 1) + fib-sect(n - 2).
```

The parameterized section declares one numeric argument and returns a numeric value. In this case, argument type is declared using the TYPEDEF syntax. The section is recursive, as the code invokes itself. The output of the program is “55”, shown in the DISPLAY statement.

Parameterized sections support optional parameters by including a default value in the argument declaration. For example, the following code:

```
repository.
    class jint as "java.lang.Integer".
...
    perform add-num(var1, var2)
    perform add-num(3, 4)
    perform add-num(3)
    perform add-num()
    ...
add-num section (p1 as jint = 1, p2 as jint = 2).
    compute tot = tot + p1 + p2
    .
```

declares a section named add-num that can be executed by passing the 2 expected parameters, but it can also be invoked with a single argument – and the second argument is assigned the default value 2, or with no parameters, and the two parameters are automatically assigned the default 1 and 2 value respectively.

Ternary operator

A ternary operator is a conditional operator in programming that takes three operands and evaluates a result based on a condition. It is commonly represented as "condition? expression1: expression2" and is supported in in many programming languages such as Java, C, C++, C#, … It is basically a short version of an assignment executed within an if ... else statement. The first term is the condition, the second term is the value returned if the condition evaluates to true, and the third term is returned if the condition evaluates to false.

Starting from version 2026 R2, isCOBOL supports the ternary operator. The syntax makes the code much shorter and more readable, and code such as the following:

```
if w-age >= 18
   set w-status to "Adult"
else
   set w-status to "Minor"
end-if
```

can be rewritten as:

```
set w-status to !! w-age >= 18 ? "Adult" : "Minor"
```

or:

```
set w-status to eval w-age >= 18 true "Adult" false "Minor"
```

The first syntax, using " !! ? :", is shorter and resembles the syntax used in other languages, while the second syntax that uses "eval true false" is more descriptive for better readability.

OOP syntax improvements

Several improvements have been introduced to the OOP syntax, including:

- support for IS INSTANCE OF condition
- support for SELF syntax in RETURNING phrase
- new RESOURCE clause in TRY statement

The INSTANCE OF condition determines whether an object reference is an instance of a particular class or interface. Using “IS INSTANCE OF” in the conditions helps developers prevent runtime exceptions by verifying an object type before casting, avoiding errors like ClassCastException. It also handles null values by automatically returning false if the object is null, preventing NullPointerException errors. The new syntax is shown in this code snippet:

```
class JString as "java.lang.String"
...
if obj is instance of JString
   ...
```

The way to return the object of self-instance from a method is now simplified using the SELF syntax in the METHOD-ID RETURNIG clause. The following snippet:

```
id division. method-id. methodName as "methodName" (str as jstring).
procedure division returning self.
   ...
end method.
```

is equivalent to the longer code that was necessary in previous releases:

```
working-storage section.
77  selfInstance object reference myClassName.
...
id division. method-id. new as "new".
procedure division.
   set selfInstance to self.
end method.
id division. method-id. methodName as "methodName" (str as jstring).
procedure division returning selfInstance.
  ...
end method.
```

Now it is no longer necessary to store an object reference that contains the self-instance in the constructor method.

The RESOURCE syntax used in the TRY statement automatically manages resources that need to be closed explicitly, such as files and database connections. As a result, the program no longer needs to handle closing these resources since they are automatically closed when the try statement completes. This feature works with any object that

implements the AutoCloseable or Closeable interface. The resulting code is cleaner and more readable, and there is no need to write explicit or nested FINALLY blocks.

The following code snippet automatically closes both the FileReader and BufferedReader objects, eliminating the need for explicit resource-cleanup code:

```
class filereader     as "java.io.FileReader"
class bufferedreader as "java.io.BufferedReader"
class ioexception    as "java.io.IOException"
...
    try resource
        fr as filereader = filereader:>new(filePath)
        br as bufferedreader = bufferedreader:>new(fr)
        declare ln as jstring = br:>readLine
        perform until ln = null
           set ln to br:>readLine
           ...
        end-perform
    catch ioexception
     ...
    end-try
```

Other enhancements

For compatibility with other COBOL dialects, such as Micro Focus, the Z"string" syntax is now supported for producing a null-terminated string, which is especially useful when calling C native functions. For example, code like this:

```
move z"abc" to par-x
```

is equivalent to:

```
move "abc" & x"00" to par-x
```

A new compiler directive, WARNING, is now supported to return a warning during compilation. This alerts developers to conditions that need attention without stopping the compile process, unlike the ERROR directive that halts compilation.

This code snippet:

```
>>DEFINE MYFLAG1 AS 1
>>DEFINE MYFLAG2 AS 3
...
>>IF MYFLAG1 = 1
      ...
>>ELSE
    >>ERROR "MYFLAG1 NOT = 1"
>>END-IF
...
>>IF MYFLAG2 = 2
      ...
>>ELSE
    >>WARNING "MYFLAG2 NOT = 2"
>>END-IF
```

when compiled produces a warning if the MYFLAG2 defined compiler constant is not set to 2. Because the top of the source is setting this constant to 3, this warning is returned when compiling:

```
--W: #112 User defined error: "MYFLAG NOT = 2";file=prog.cbl...
```

Since the condition is triggering a WARNING, the compiled .class is still generated.

isCOBOL Runtime

The 2026 R2 release features improved performance in several areas, many of which can be gained by simply recompiling existing source code.

ESQL now supports intercepting SQL statements for logging or tuning, and new configuration settings and library routines have been introduced.

Better performance

The latest isCOBOL runtime can execute programs compiled with previous releases of the isCOBOL compiler, easing the burden of upgrading programs when updating the runtime in a production environment, since recompilation is not required. However, recompiling programs using the 2026 R2 release produces faster executables, since the output .class files are better optimized. This is useful for batch programs or GUI programs that perform lengthy operations, resulting in quicker runtimes.

This release includes several optimizations in key areas:

- SEARCH statement and comparison statements like IF to compare the value of occurs data items in fixed OCCURS tables and tables with DEPENDING ON syntax, especially when the index used in the loop is the one implicitly declared in the INDEXED BY clause. An example of this is:

```
03  w-table occurs 1 to 100000 times depending on w-max-occ
                               ascending w-name indexed by w-idx.
    ...
77  my-idx pic 9(7) comp-3.

...
search w-table
  at end exit paragraph
when search-name = w-name(w-idx)
   ...
end-search
perform varying w-idx from 1 by 1 until w-idx > w-max-occ
   if search-name = w-name(w-idx)
      ...
   end-if
end-perform
perform varying my-idx from 1 by 1 until my-idx > w-max-occ
   if search-name = w-name(my-idx)
      ...
   end-if
end-perform
```

- INITIALIZE statements on structures that have a fixed number of OCCURS, like in the following code snippet:

```
01 ws-table.
   05 ws-row occurs 500.
    10 ws-col occurs 100.
       15 ws-cod  pic x(9).
       15 ws-name pic x(20).

...
initialize ws-table
```

- INVOKE statement that executes a METHOD-ID with just one paragraph in a CLASS-ID source. In this case, the sub class is no longer generated, improving execution performance. The code snippet for this is:

```
myObj::setCod(w-cod)
myObj::setName(w-name)
set w-cod  to myObj::getCod()
set w-name to myObj::getName()
```

A table of performance gains is shown in Figure 12, *Performances comparison table*, where the comparison is made between isCOBOL 2026R2 runtime without recompiling and isCOBOL 2026R2 after recompiling programs. The test was run in Windows 11 Pro 23H2 13th Gen Intel(R) Core(TM) i7-1355U 1.70 GHz with 32GB of RAM, using Oracle Java 64-Bit Server version 17.0.8+9-LTS-211. All times are in seconds. The number of iterations in the loop is shown in the image, since it differs depending on the test.

Figure 12. Performances comparison table.

Statement	Num times	Previous rel	2026 R2
SEARCH statement on the table using the implicit INDEXED BY field	1 M	96,07	16,13
IF condition on occurs using the implicit INDEXED BY field	1 M	91,69	8,68
IF condition on occurs using as index a PIC 9(9) COMP field	1 M	56,79	24,37
INITIALIZE on 01 level with fixed length OCCURS containing numeric fields:	20 K	48,09	7,14
INVOKE some method-id with only a paragraph to set/get fields	20 M	13,12	0,74
Total Time:		**305,76**	**57,06**

Intercepting ESQL statements

It is now possible to intercept ESQL statements with a hook class for logging or tuning purposes. The hook class is associated with the runtime session using the following configuration property:

```
iscobol.esql.log.performance_monitor=<classname>
```

The hook class must implement the com.iscobol.rts.EsqlCobolStatementTracer interface, which defines the following three methods:

- Trace: Invoked for every ESQL statement. It receives the execution timestamp, the SQL statement, the query parameters, and the execution time.
- EnterProgram: Invoked when execution enters a program. It receives the program name.
- ExitProgram: Invoked when execution exits a program. It receives the program name.

This feature is especially useful for query logging and performance tuning. It allows you to log all ESQL statements executed during a runtime session and identify the most expensive queries. For example, the java source provided on the next page is a simple implementation that prints the collected information to the standard output.

When running, the following configuration option needs to be loaded:

```
iscobol.esql.log.performance_monitor=EsqlTrace
```

```
import java.sql.SQLException;
import java.util.Vector;
import java.util.Enumeration;
import com.iscobol.types.CobolVar;
import com.iscobol.rts.EsqlCobolStatementTracer;
import com.iscobol.rts.EsqlHostVar;

public class EsqlTrace implements EsqlCobolStatementTracer {
   public void trace(long timestamp, long nanoseconds,
          String normalizedSQL, Vector parameters, SQLException sqlEx) {
       System.out.println("Timestamp: " + timestamp);
       System.out.println("Time spent (ns): " + nanoseconds);
       System.out.println("Query: " + normalizedSQL);
       try {
          StringBuilder sb = new StringBuilder();
          if (parameters.size() == 0) {
             System.out.println("Parameters: empty vector!");
          } else {
             Enumeration vEnum = parameters.elements();
             while(vEnum.hasMoreElements()) {
                Object ne = vEnum.nextElement();
                if (ne != null) {
                   EsqlHostVar ehv = (EsqlHostVar)ne;
                   CobolVar cv = ehv.hostVar;
                   if (cv.isNumeric()) {
                      sb.append(", "+cv.toint());
                   }           else {
                      sb.append(", '"+cv.toString()+"'");
                   }
                }
            }
               System.out.println("Parameters: " + sb);
            }
       } catch (Exception e) {
           e.printStackTrace();
       }
       if (sqlEx != null) {
           System.out.println("Exception: " + sqlEx);
       }
   }

   public void enterProgram(String name) {
       System.out.println("enterProgram[" + name + "]");
   }

   public void exitProgram(String name) {
       System.out.println("exitProgram[" + name + "]");
   }
}
```

Other improvements

A new routine, named J$DUMP, is provided to programmatically create on-demand heap or thread dumps, storing the content in a file on the disk.

A heap dump is a snapshot of the Java Virtual Machine's heap memory at a specific point in time. It captures exactly what objects exist in memory, how much space they consume, and how they reference one another.

A thread dump is a text-based snapshot of the execution state of all active threads running inside a Java Virtual Machine at a specific moment in time.

Developers primarily use them as a diagnostic blueprint to investigate application freezes, memory leaks, or performance issues. In previous releases external commands or utilities were needed to create dump files, but it is now possible to add a CALL directly to the COBOL source code when necessary.

The following code snippet:

```
call "J$DUMP" using jdump-heap, "filename.hprof"
call "J$DUMP" using jdump-thread, "filename.tdump"
```

calls the new routine to store two files, containing the heap and thread dumps respectively.

The dumps can be fed to tools such JVisualVM or Eclipse MAT for analysis.

A new property, named iscobol.file.index.encrypt.type is available to specify the c-tree RTG encryption algorithm. Using the **ctreej** file handler, developers can choose the encryption algorithm that c-tree must use when iscobol.file.index.encrypt is set to true or when WITH ENCRYPTION is specified in the FILE CONTROL definition.

In previous versions, CAMO was used. The following algorithms are now supported:

- DES8, DES16, DES24
- TWOFISH16, TWOFISH24, TWOFISH32
- BLOWFISH8, BLOWFISH9, etc. up to BLOWFISH56
- AES16, AES24, AES32

CAMO is still used as default when iscobol.file.index.encrypt.type is not set.

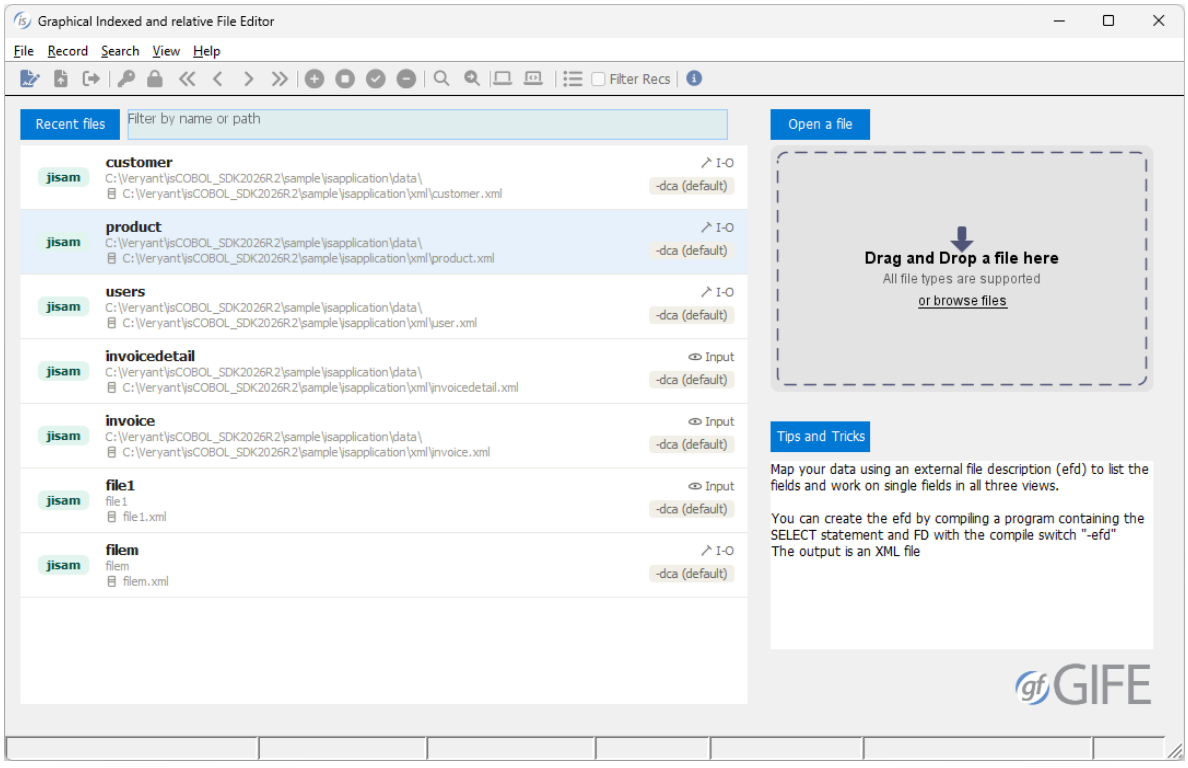
IsCOBOL utilities

IsCOBOL 2026 R2 contains a new version of GIFE, the tool that lets you view and update records in indexed and relative files. The new release introduces an initial window that simplifies the open operation, and the Byte view now uses a more efficient component. GIFE and the isCOBOL Server Panel now support the dark mode.

GIFE initial window

GIFE shows a new initial screen for easier selection of recent files and a DropZone component that supports files drag and drop operations. The new window is shown in Figure 13, *GIFE initial window*.

Figure 13. GIFE initial window.

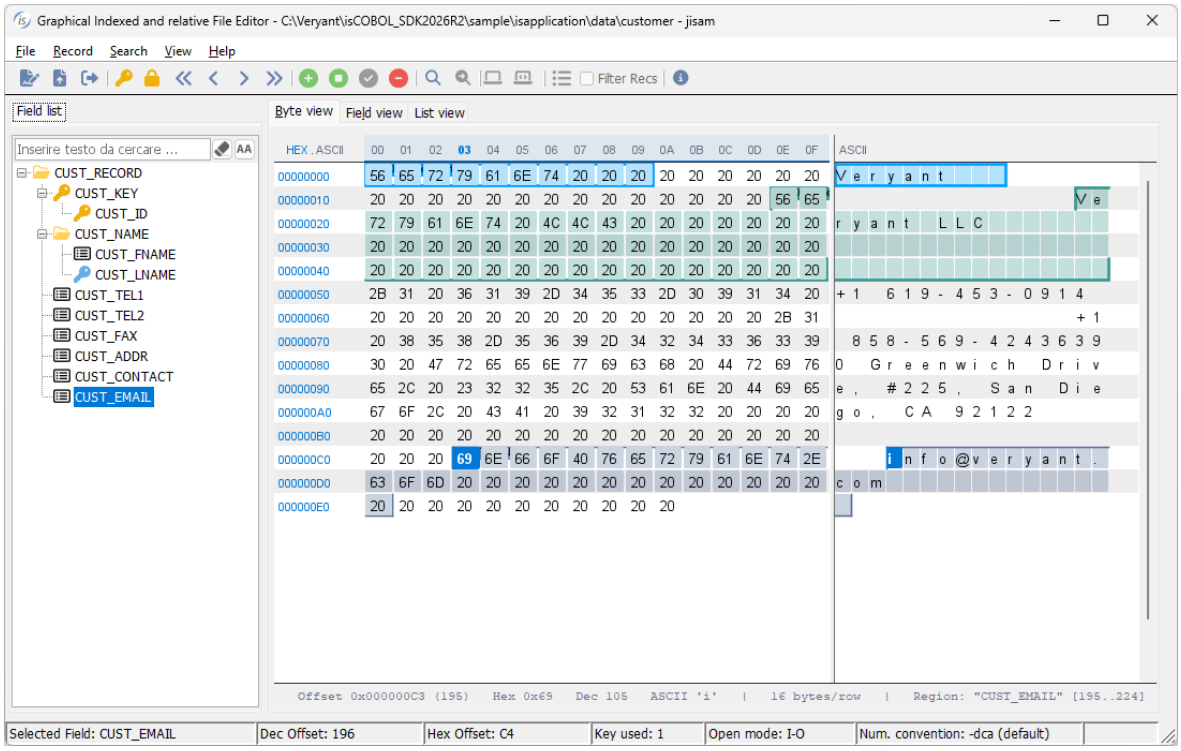


GIFE Byte view

The Byte view component has been upgraded to improve the Hex/ASCII Editor in the Byte View. It also supports a split divider between Hex and ASCII views to let you resize the two areas. It also improves support for high screen resolution by providing improved rules for window resizing or for maximized windows. The bottom section of the view shows the offset and size of the region selected in the record.

In Figure 14, *GIFE byte view*, the new component is shown.

Figure 14. GIFE byte view.



Dark mode support

Since many developers prefer dark mode when working with the isCOBOL IDE, Microsoft Visual Studio Code, or while debugging, the new versions of the most used isCOBOL Utilities now support dark mode as well. In this release, GIFE and the isCOBOL Server Panel can be used in dark mode.

In Figure 15, *GIFE dark mode* shows the dark mode variant of the GIFE utility, which can be activated by running the command:

```
iscrun --flatlaf FlatDarkLaf -utility GIFE
```

Figure 15. GIFE dark mode.

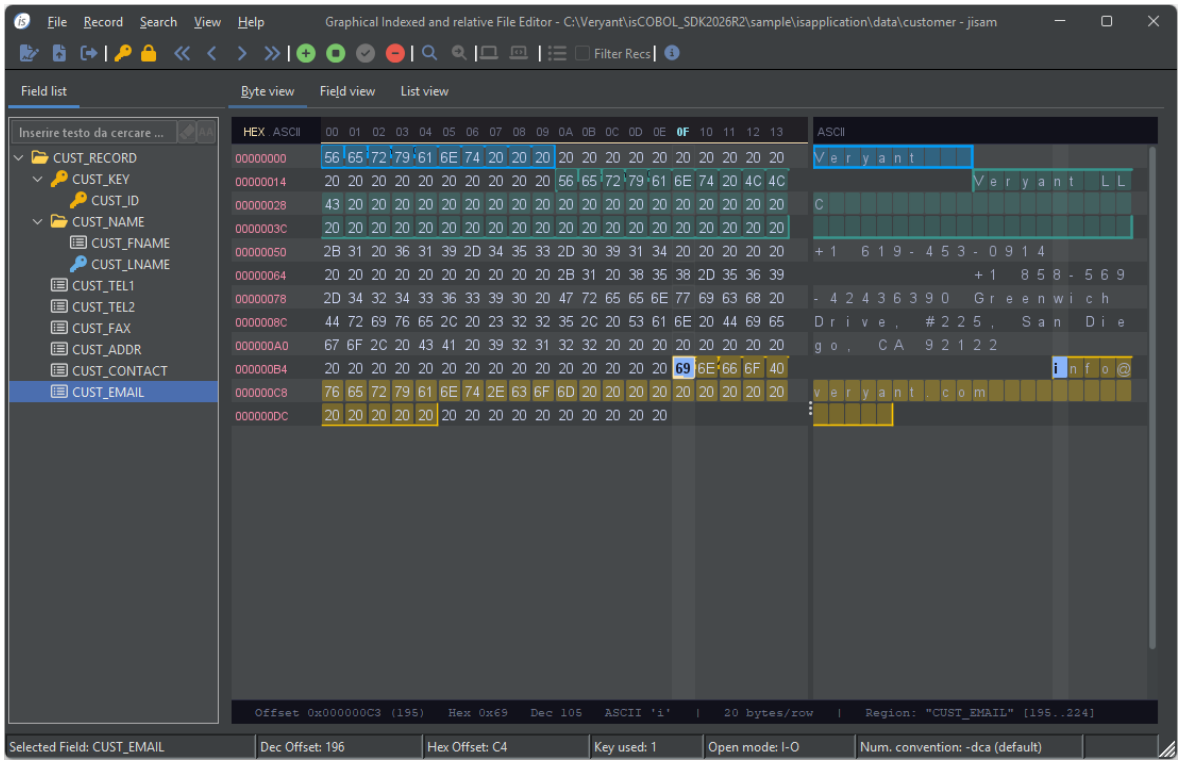


Figure 16, *Panel dark mode*, shows the isCOBOL Server Panel running in dark mode using the command:

```
iscclient --flatlaf FlatDarkLaf -user admin -password admin -panel
```

Figure 16. Panel dark mode.