isCOBOL™ Evolve

isCOBOL Evolve 2019 Release 2 Overview

isCOBOL Evolve 2019 Release 2 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL™ Evolve, isCOBOL Evolve 2019 R2.

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

The 2019R2 release has many new features for GUIs, such as a new zoom layout-manager, new borders and color management, and features that let developers modernize applications with little or no effort.

isCOBOL 2019R2 has increased performance for CALL statements and C-Tree file handling.

The EIS suite has been upgraded, giving more power to developers and more control over application deployed with WebClient.

Details on these enhancements and updates are included below.

GUI enhancements

Many improvements to GUI controls have been implemented in this release, such as a new Layout Manager to easily manage scaling GUIs, custom borders that can now be set on controls, more color choices to enrich GUIs, and a new hook for mouseovers..

Zoom Layout Manager

Starting with isCOBOL 2019R2, all the isCOBOL GUI, Graphical windows, Screen Programs or isCOBOL WOW programs generated by IDE can take advantage of an easy to use and low impact layout manager to handle application resizing.

By setting the configuration setting:

```
iscobol.gui.layout_manager=lm-zoom
```

the new Zoom Layout manager is activated, windows automatically become resizable, and all controls are adjusted in size when increasing the window width, and in font size when increasing the window height. This behavior is completely automatic, requiring no effort from developers. All that is needed to enable this behavior is to set a configuration property.

This helps quickly and easily to solve the resizing issues of running applications on a variety of monitors with different sizes and resolutions, with zero code changes.

Individual windows can be targeted by enabling the LM-ZOOM layout manager in the display window statement, as shown in the code below.

```
77 zoom-layout handle of layout-manager, lm-zoom.

display standard graphical window resizable
        layout-manager zoom-layout
```

When migrating programs from COBOL-WOW, the LM-ZOOM Layout Manager can be enabled on specific forms by setting the new `layoutManager` property in the isCOBOL IDE WOW Painter.

Figure 1, *Default size,* shows how the program runs at startup, before the user resizes the window.

Figure 2, *Window stretched horizontally,* shows how the GUI looks like after the user stretches the window horizontally, and Figure 3, *Window fully resized,* shows how the GUI reacts after the window is resized both horizontally and vertically.

Figure 1. Default size

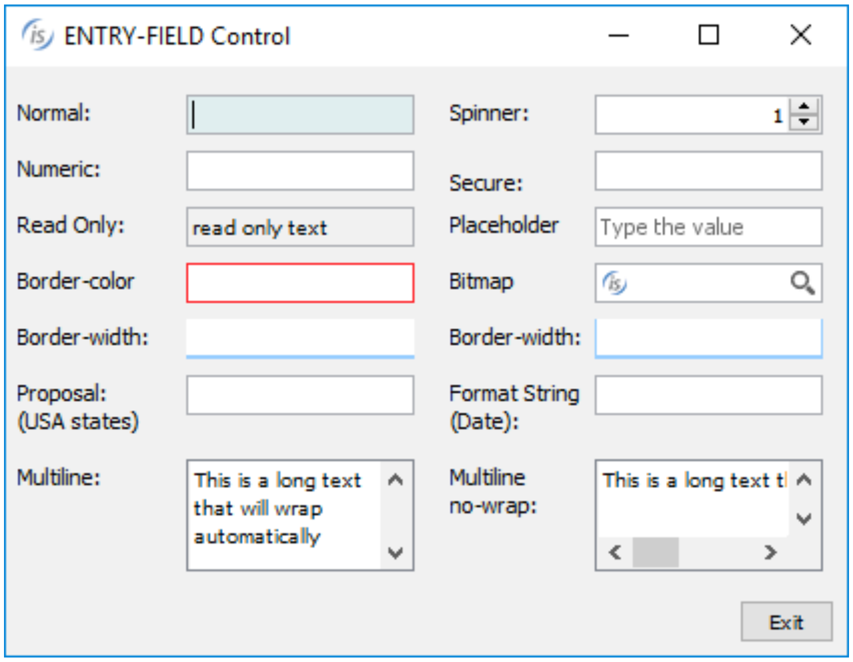
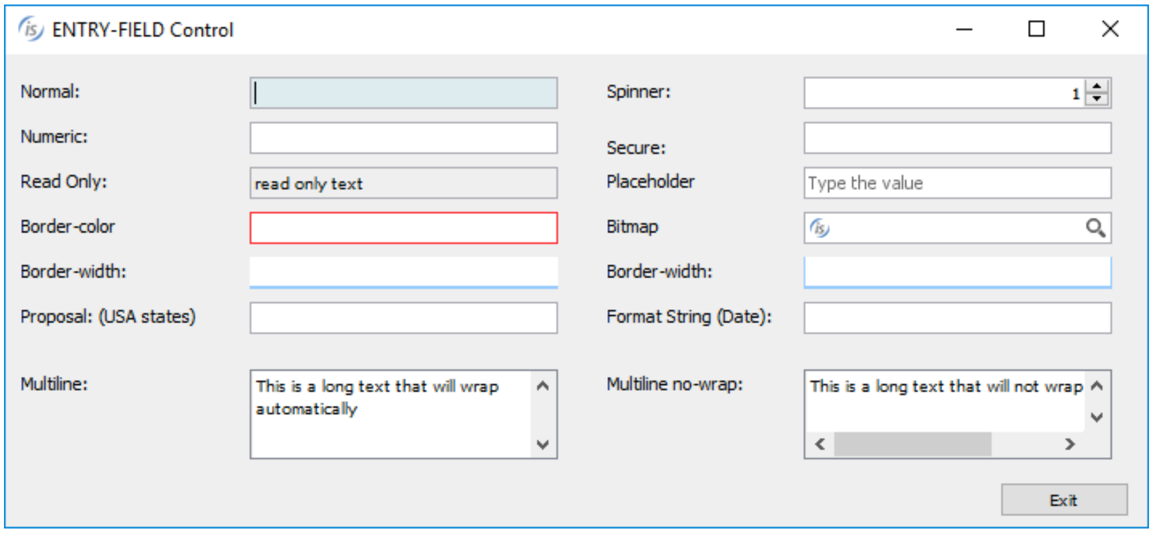


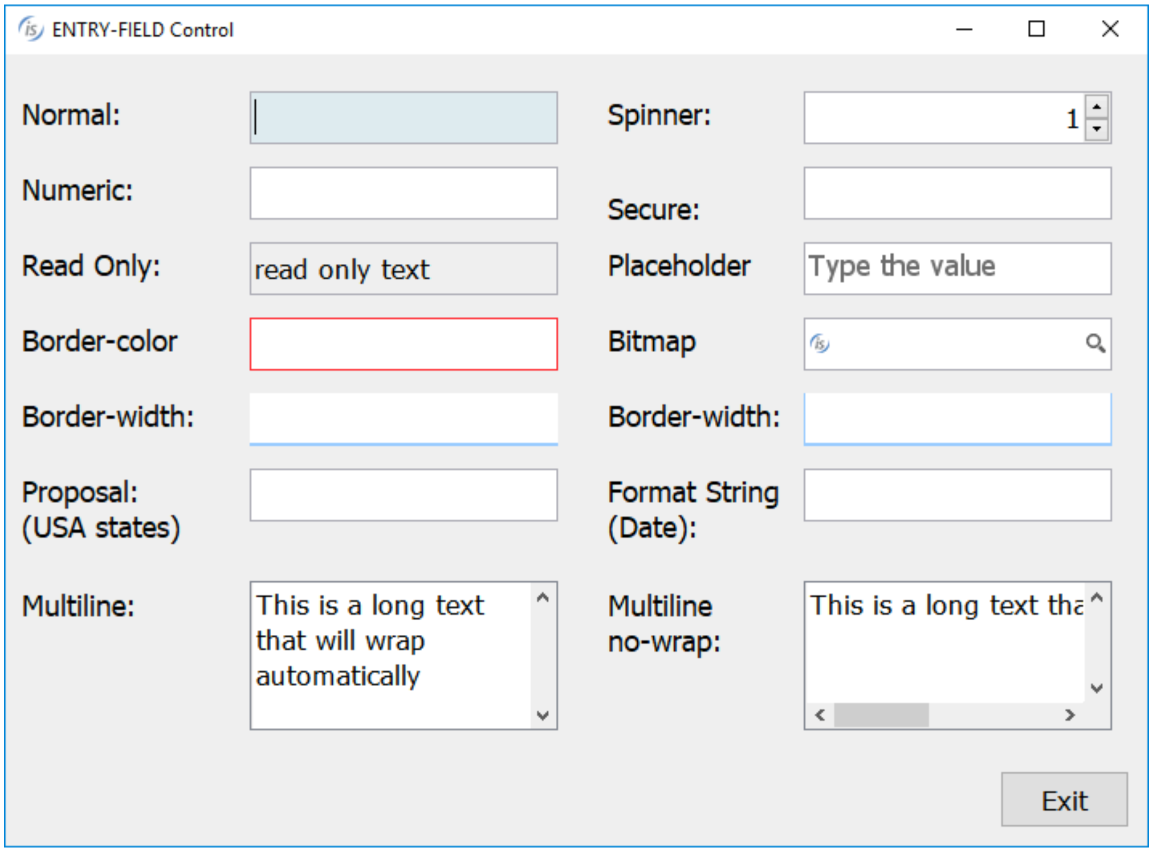
Figure 2. Window stretched horizontally



ENTRY-FIELD Control

Normal:		Spinner:	1
Numeric:		Secure:	
Read Only:	read only text	Placeholder	Type the value
Border-color		Bitmap	
Border-width:		Border-width:	
Proposal: (USA states)		Format String (Date):	
Multiline:	This is a long text that will wrap automatically	Multiline no-wrap:	This is a long text that will not wrap

Figure 3. Window fully resized



ENTRY-FIELD Control

Normal:		Spinner:	1
Numeric:		Secure:	
Read Only:	read only text	Placeholder	Type the value
Border-color		Bitmap	
Border-width:		Border-width:	
Proposal: (USA states)		Format String (Date):	
Multiline:	This is a long text that will wrap automatically	Multiline no-wrap:	This is a long text tha

Custom borders

Controls have always had a border-color property that allows custom border colors to be applied. A new property, `border-width`, now allows you to control how borders are created on selected controls. The property is an array of 4 integers that specify the width in pixels of the top, left, bottom and right side of the control.

For example, by setting the following property of an entry-field

```
border-width (2, 2, 2, 2)
```

the control will have 2 pixels-width borders on all sides. Values can be combined to give the application a more modern look, or to make specific controls stand out from others. For example, by setting the following borders on an entry-field

```
border-width (0, 0, 1, 0)
```

only the bottom border will be activated, resulting in a control that looks much like Google's material design guidelines input elements.

Using the values:

```
border-width (0, 1, 1, 1)
```

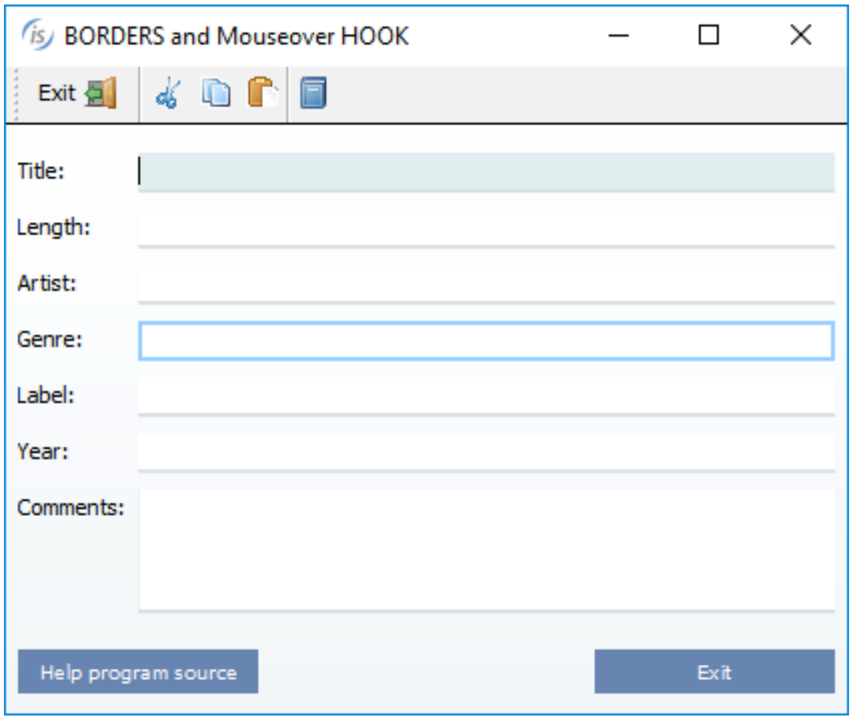
a "basket-like" effect can be achieved.

These styles can be combined with new additional configuration settings to generate different border-widths and have them applied automatically when controls gain focus (`iscobol.gui.curr_border_width`) or when the mouse hovers over controls (`iscobol.gui.rollover_border_width`).

See more details in the "New configuration properties" paragraph in the Framework improvements section below.

Figure 4, E*ntry-field borders,* shows a modern looking GUI using the `border-width` property to set the default border, the currently focused control border, and the border style for controls being hovered by the mouse.

Figure 4. Entry-field borders



Additional color settings

New properties have been added to set different colors for specific items or the selected item in controls that handle items, such as tree-views, list-boxes and combo-boxes. Combining all the new properties will ease the effort of modernizing existing GUI applications.

The code snippet below shows how to set a specific color for the selected item in the tree-view declared in the screen section:

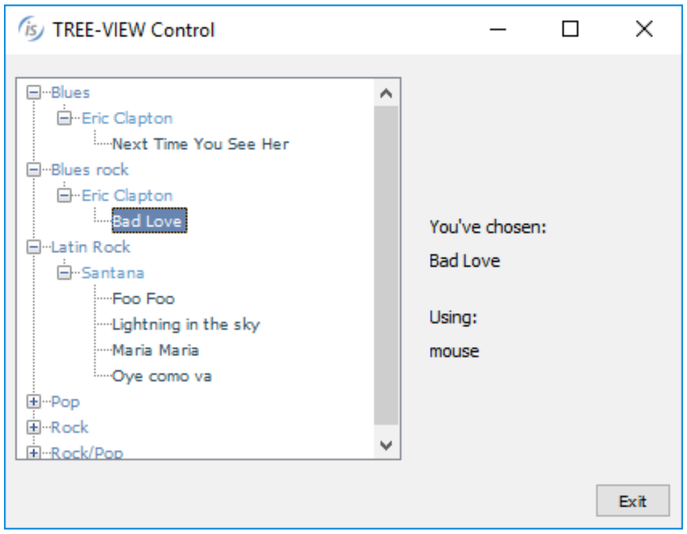
```
03 Tv1 tree-view
   selection-color 483
   ...
```

The code snipped below shows how to set colors on a specific item added to a tree-view using code run at runtime:

```
modify Tv1 item-to-add "item1"
           item-color 5
modify Tv1 item-to-add "item2"
           item-foreground-color rgb x#FF0000
           item-background-color rgb x#84FFFF
```

Figure 5, “tree-view colors”, shows how to enhance the looks for tree-views using different colors on different item levels.

Figure 5. tree-view colors



You've chosen:
Bad Love

Using:
mouse

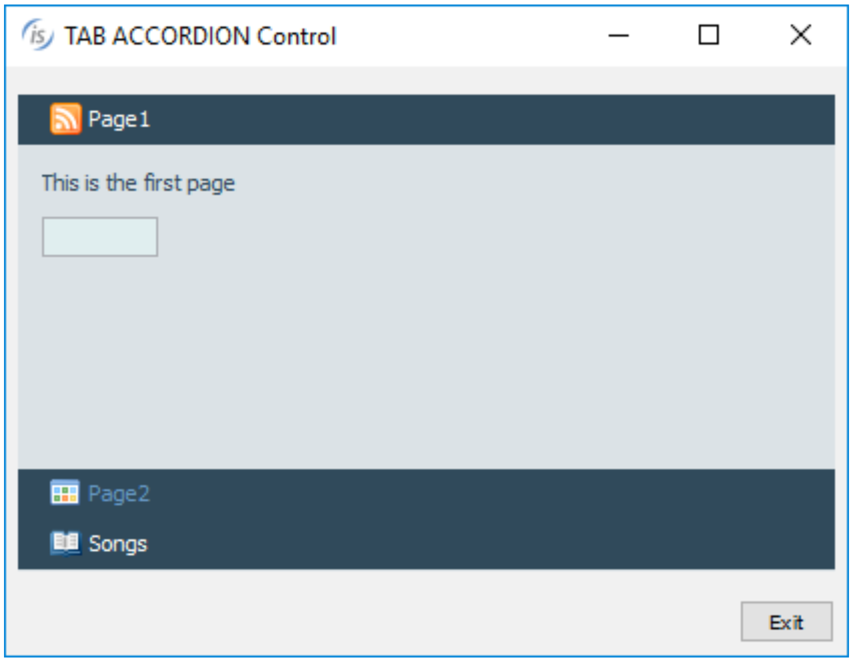
Accordion tab-controls also support the new properties and styles to enhance the color selection, allowing a specific color to be set for the title tag area and the color to be applied when the mouse hovers over the control. Additionally, the new style **tab-flat** will cause the tab title to be rendered with a flat style. Figure 6, *Accordion colors,* shows the effect of setting colors with the tab-flat style.

The **tab-delay** property allows you to set the duration of the animation effect displayed when the user selects a different tab, resulting in a smooth transition effect.

The code snippet below configures additional colors in the accordion declaration in screen section:

```
 03 Acc1 tab-control accordion
         tab-flat
         background-color rgb 145
         foreground-color rgb 10745433
         tab-background-color rgb 145
         tab-foreground-color rgb 10745433
         tab-rollover-color rgb 16711680
         tab-delay 2000
         ...
```

Figure 6. Accordion colors



Push-button controls with the flat property set to true will now render all configured colors, no matter what LAF is active at runtime.

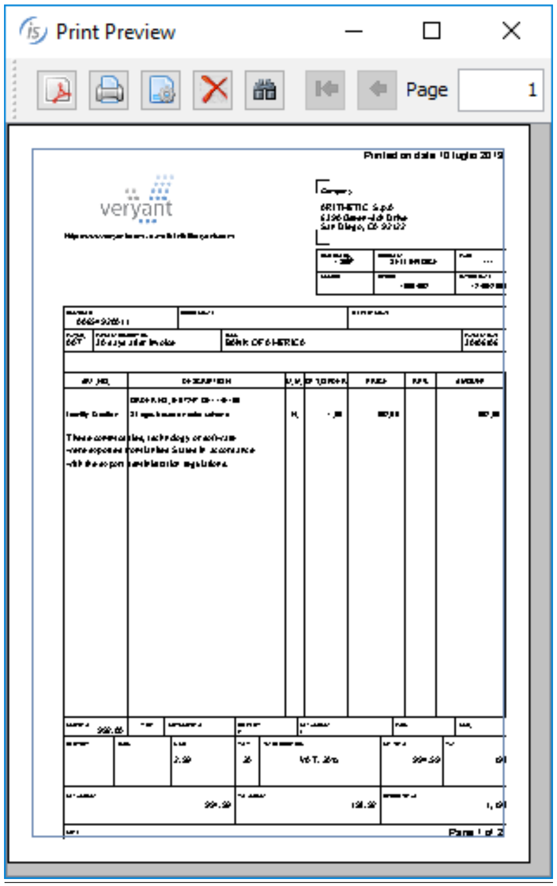
The printer preview dialog now allows you to set the color of the Printable Area Box. To read and write the color, two new methods class have been implemented in the com.iscobol.rts.print.SpoolPrinter class.

```
java.awt.Color getPrintableAreaBoxColor()

setPrintableAreaBoxColor(java.awt.Color)
```

Figure 7, *Print preview margin color,* shows the result of setting a different color for the preview printable area box.

Figure 7. Print preview margin color



Hook on mouseover

Traditionally, there has been a single way to execute programs using a function key when a specific control has focus. This has usually been exploited to provide contextual help on specific controls, typically using the F1 key. The exception value to be used can be set with the following code:

```
SET EXCEPTION VALUE 1 TO ITEM-HELP
```

The following configuration variable is used to configure the program to run when the function key is pressed:

```
iscobol.help_program=myhelp
```

The program being run (in this example: myhelp), receives the information needed to determine which control has focus in the linkage section, as well as relative actions to be taken.

Now, the same functionality can be obtained by hovering the mouse on a control, without the user needing further interactions. A new property is available to set the delay from the moment the mouse stops and when the program is invoked.

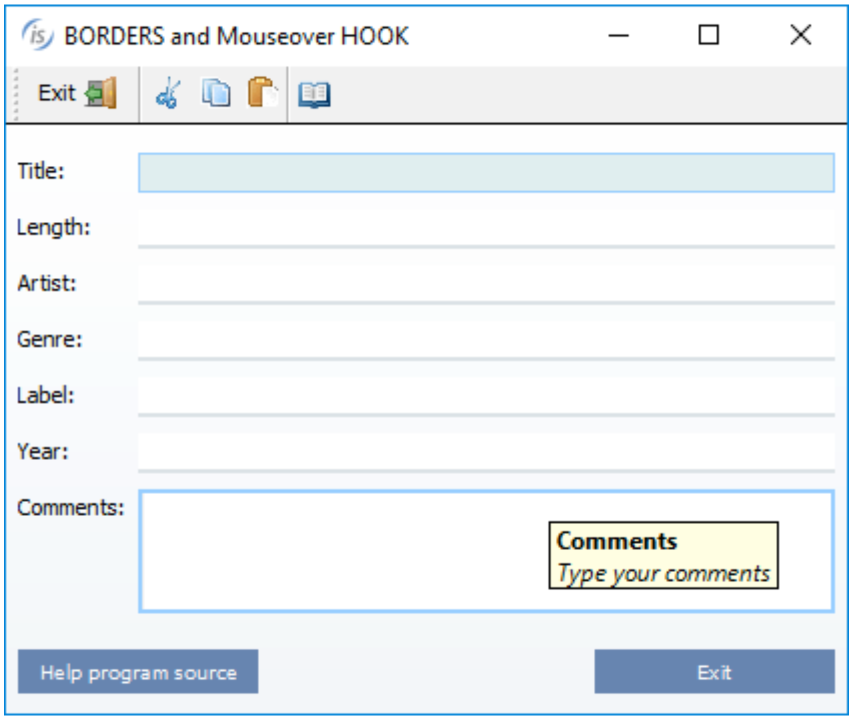
```
iscobol.help_program_mouse_stop_delay=n
```

The called program has access, in the linkage section, to the following information:

- the control's handle
- the control’s ID
- the control’s help ID
- the handle of the control’s owning window

Figure 8, *Hint on hook*, shows the result of programmatically showing a hint when the configured program is run. Also shown is the flat push-button with background-color.

Figure 8. Hint on hook



Framework improvements

Performance of CALL statements and operations on c-treeRTG indexed files have been greatly improved. New configuration properties and new library routines have been introduced.

Performance of CALL statements

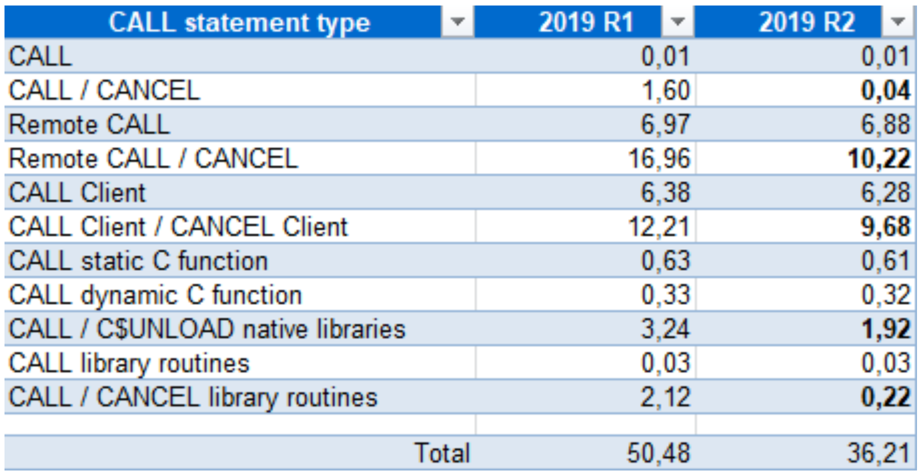
Performance of CALL statements has been improved across the board, especially when using CALL/CANCEL (or C$UNLOAD_NATIVE for native libraries) statements, under all circumstances (CLASSPATH or code_prefix). This is an additional improvement, adding to the optimizations already introduced in 2019 R1 edition, which was limited to the COBOL calls under code_prefix.

The gain has been achieved by saving the information regarding the type of the executed CALL, whether it is a COBOL local call, a COBOL remote call, a library routine call (such as any of the C$*, CBL_*, P$*, WIN$* calls), or a Native Dynamic call or Native Static call for C functions. In a later call to the same function, the runtime already has access to its type, saving the time needed to determine it once again.

A table of performance gains is shown in Figure 9*, CALL performance improvements,* which shows a performance comparison between isCOBOL 2019R1 and isCOBOL 2019R2. The tests were run in Windows 10 64-bit on an Intel Core i5 Processor 4440+ clocked at 3.10 GHz with 8GB of RAM, using Oracle JDK1.8.0_211. All times are in seconds.

The called COBOL program contains a linkage group data item with 50 child items. The number of CALL/CANCEL iterations is 10,000 for the COBOL program and 100,000 for Native calls and Library routines using a single parameter.

Figure 9. CALL performance improvements



CALL statement type	2019 R1	2019 R2
CALL	0,01	0,01
CALL / CANCEL	1,60	**0,04**
Remote CALL	6,97	6,88
Remote CALL / CANCEL	16,96	**10,22**
CALL Client	6,38	6,28
CALL Client / CANCEL Client	12,21	**9,68**
CALL static C function	0,63	0,61
CALL dynamic C function	0,33	0,32
CALL / C$UNLOAD native libraries	3,24	**1,92**
CALL library routines	0,03	0,03
CALL / CANCEL library routines	2,12	**0,22**
Total	50,48	36,21

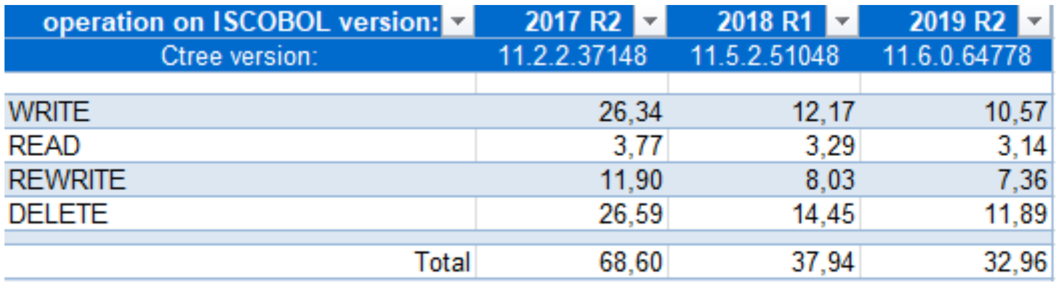
Performance of c-treeRTG indexed files

Performance of c-treeRTG indexed files has been improved constantly with a synergy between isCOBOL and the c-treeRTG file system. All COBOL statements that access indexed files (such as WRITE, REWRITE, DELETE, READ) have been improved. To achieve the maximum performance gain it is strongly recommended that you upgrade all your Veryant products, especially when older releases are still being used in production environments, as new features, performance improvements, and tweaks, are constantly being added to the entire suite.

A table of performance gains is shown in Figure 10, c-treeRTG performance improvements, and shows a performance comparison between different isCOBOL versions (2017R2, 2018R1 and isCOBOL 2019R2) with the embedded OEM c-treeRTG release available for each release (v.11.2, v11.5 and the latest v.11.6). The tests were run in the same environment and using the same hardware as the previous test. All times are in seconds.

The program used for the test is IO_INDEXED.cbl, which is installed under sample\io-performances. The number of records used for this test is 500,000. The test is executed under the default c-treeRTG server configuration, and the property iscobol.file.index=ctreej is set in isCOBOL configuration to access c-tree files. It's important to know that the c-treeRTG file system offers specific configuration settings both client-side and server-side that help in tuning the performances, depending on the architecture and needs, allowing further performance gains.

Figure 10. c-treeRTG performance improvements



operation on ISCOBOL version:	2017 R2	2018 R1	2019 R2
Ctree version:	11.2.2.37148	11.5.2.51048	11.6.0.64778
WRITE	26,34	12,17	10,57
READ	3,77	3,29	3,14
REWRITE	11,90	8,03	7,36
DELETE	26,59	14,45	11,89
Total	68,60	37,94	32,96

New configuration properties

Several new configuration properties have been implemented to enhance character based applications:

- `iscobol.terminal.cursor_blink=n` to set the blinking cursor speed in character accepts, in milliseconds
- `iscobol.terminal.cursor_color=n` to specify the cursor color number (0-15) in character accepts (default -1)
- new value in keystroke configuration: `edit=erase-all` used to clear all the fields in the character screen section in accepts

New configuration properties to enhance the graphical based application:

- `iscobol.gui.curr_border_color=n` to set the border-color of the currently focused entry-field
- `iscobol.gui.curr_border_width=n1 n2 n3 n4` to set the border-width of the currently focused entry-field
- `iscobol.gui.rollover_border_color=n` to set the border-color of entry-fields when the mouse hovers over them
- `iscobol.gui.rollover_border_width=n1 n2 n3 n4` to set the border-color of entry-fields when the mouse hovers over them
- `iscobol.gui.window_title=xxx` to set the default window title when none is set in the COBOL source
- `iscobol.help_program_mouse_stop_delay=n` to set the time between when the mouse stops over a control and when the configured program is run, when using the "mouseover hook" feature. This can be useful, for example, to manage contextual help systems when existing COBOL applications need to be modernized without code changes.

Additional configuration properties:

- `iscobol.auto_input_mode=true` to activate IME (Input Method Editor) when the focus is on a field associated to a PIC N data-item. This works on both character and graphical user interface and it's useful when the application needs to input Chinese, Japanese, Korean and Indic characters
- `iscobol.key.accepted_control_characters=\u{hex value}` to specify control characters that should not be discarded during editing. This is useful for applications that use "barcode readers/scanners/guns" to insert values in the COBOL application. Multiple values can be specified by separating each with commas.

New ISUPDATER configuration properties:

- `swupdater.new_jvm_always=true` to always run the mainClass, defined in the `swupdater.mainclass` property, in a new JVM, even if there is no need to update any component on the machine
- `swupdater.jvm_options=<java-options>` to specify the Java options for the newly started JVM after updating components, if the `new_jvm_always` property is set to false (or is not set), or always if the setting is set to true. If `new_jvm_always` is not set, the same options set on the invoking JVM are used on the newly instantiated one.

isCOBOL Compiler

Starting from isCOBOL 2019R2, dynamic variables (X ANY LENGTH and OCCURS DYNAMIC) have been improved. New compiler configurations have been added to inject code for GUI controls and a new compatibility compiler option has been implemented.

<u>Dynamic variables</u>

Dynamic variables allow developers to declare variables and arrays without knowing the size beforehand. One obvious advantage, on complex group level definitions, is to lower the memory footprint of the application. Typically, COBOL programmers have to choose a maximum value for an OCCURS array, which is most likely a random "big enough" value to handle each case. The maximum number of items assigned for the array may never be needed, but memory is assigned nonetheless.

Declaring the array as OCCURS DYNAMIC will allow dynamic sizing of the array, optimizing memory consumption.

The same can be applied to a PIC X(...) or PIC N(…) variable, where typically a maximum size needs to be determined. By using PIC X ANY LENGTH or PIC N ANY LENGTH, memory usage is optimized by allocating the right amount of memory needed to hold the contents.

Starting with isCOBOL 2019R2, group level variables containing dynamic items can be moved or compared to other compatible group level variables. Before this release, moving or comparing group level variables containing dynamic items were limited to non-dynamic items only.

For example, the following syntax defines 2 identical structures that contain dynamic child data items:

```
 WORKING-STORAGE SECTION.
01 group1       group-dynamic.
   03 g1v1      pic x.
   03 g1v2      occurs dynamic capacity k-g1v2.
      05 g1v2a pic x.
      05 g1v2b pic 9.
   03 g1v3      pic x any length.
01 group2.
   03 g2v1      pic x.
   03 g2v2      occurs dynamic capacity k-g2v2.
      05 g2v2a pic x.
      05 g2v2b pic 9.
   03 g2v3      pic x any length.
```

The GROUP-DYNAMIC clause has been added to better describe how the structure is to be handled during moves and comparisons. This clause is not mandatory, and the compiler implicitly assumes it at compile time in groups that contain dynamic data items.

The advantage of this new approach is that now the following statements:

```
move group1 to group2
if group1 = group2
```

will handle the dynamic items during moves and comparisons. This only works when the two structures are mirrors of each other. This eases the code rewrite needed when moving code from static structures to dynamic ones.

Other improvements allow developers to retrieve the current capacity of an occurs dynamic variable, without the need to declare the capacity in the data division, as the code below shows.

```
set curr-size to capacity of g2v2
```

This is very useful when the programs need to retrieve different capacities in nested occurs dynamic items.

The SORT statement, which can be used to sort data in an array structure, now supports sorting data structures containing dynamic occurs data items, as shown below:

```
    sort cust-array     on descending key cust-name
                        on ascending key cust-city
```

ANY LENGTH data items can now be pre-allocated using the WITH SIZE clause on the INITIALIZE statement. This will initialize the variable with the amount of spaces defined in the with size clause.

For example

```
    initialize var-1
```

initializes var-1 with a zero-length size, while

```
    initialize var-2 with size 3
```

initializes var-2 with 3 spaces.

During the migration from static occurs to dynamic occurs inside groups, the compiler now issues a warning to identify code that is potentially affected by the changes in the handling of dynamic variables. For example, the following lines of code:

```
 display group1
if group1 = "ab1c"
```

will result in a warning being issued if group1 contains dynamic variables, to warn the developer that the statements could fail. The warning issued is as follows:

```
--W: #257 Dynamic items will be ignored: GROUP1
```

Compiler code injection

New compiler configuration settings are implemented to inject COBOL code in all controls of a specific type at compile time.

```
iscobol.compiler.gui.<control-name>.defaults=...
```

where <control-name> can be any of the following: bar bitmap, check_box, combo_box, date_entry, entry_field, frame, grid, java_bean, label, list_box, push_button, radio_button, ribbon, scroll_bar, slider, status_bar, tab_control, tree_view, web_browser, window, tool_bar.

This feature simplifies the modernization process for GUI applications and reduces developing efforts.

As an example, when compiling the following screen section controls:

```
01  s1.
   03 ef1 entry-field
        line 2 col  2 size 10.
    03 ef2 entry-field
        line 2 col 14 size 10.
    03 ef3 entry-field
        line 2 col 26 size 10.
    03 pb1 push-button
        line 5 col 10 size 10
        title "Save" exception-value 1.
```

with the following compiler configuration:

```
iscobol.compiler.gui.push_button.defaults=flat, background-color -14675438
iscobol.compiler.gui.entry_field.defaults=border-color rgb x#dae1e5, \
                                         border-width (0 0 2 0 )
```

the compiler will compile the source code as if it were written as:

```
01  s1.
   03 ef1 entry-field
        border-color rgb x#dae1e5,
        border-width (0 0 2 0)
        line 2 col  2 size 10.
 03 ef2 entry-field
        border-color rgb x#dae1e5,
        border-width (0 0 2 0)
        line 2 col 14 size 10.
 03 ef3 entry-field
        border-color rgb x#dae1e5,
        border-width (0 0 2 0)
        line 2 col 26 size 10.
 03 pb1 push-button
        flat, background-color -14675438
        line 5 col 10 size 10
        title "Save" exception-value 1.
```

Code injection also affects controls created with single display statement

```
 display push-button line 5 col 25 size 10
         title "End" exception-value 27
         handle in h-pb.
```

With code injection, an entire application can be recompiled without code changes, and changing the configuration variables can result in a completely different looking application, allowing modernization to take place without altering the source code.

Code injection works by inserting the text in the configuration variables in the source code where controls are declared or created. Syntax errors in the configuration variables will result in compilation errors.

Compatibility enhancements

A new compiler option, –cr, has been added to support new syntax, enhancing compatibility with RM/COBOL v8 or greater. This option allows us to support the following additional syntax:

- `PROGRAM-ID` and `WHEN-COMPILED` special registers
- `ACCEPT` data-item `FROM DATE-COMPILED`
- specific management for `ERASE` clause and `DISPLAY` without `LINE` for RM/COBOL compatibility

New library routines are now supported to simplify the migration from RM/COBOL:

- `C$MBAR` to display a menu bar on the active window
- `C$RBMENU` to create a pop-up menu on the active window
- `C$SBAR` to display a single panel status-bar
- `C$SCRD` to read character text
- `C$SCWR` to write character text
- `C$TBAR` to display a tool-bar on the active window
- `WOWGETWINDOWTYPE` to inquire for the window type
- `C$SHOW` to hide and show the main window

isCOBOL Remote Debugger

Starting from isCOBOL 2019R2, the isCOBOL Remote Debugger allows stopping execution only on breakpoints specifically set, instead of automatically breaking on the first statement of the first program compiled in debug mode. This is very useful in multithreaded environments, such as a J2EE container. To activate this feature, a new configuration setting needs to be set on the server side where the COBOL program is running:

```
iscobol.rundebug.auto_pause=false
```

this setting works in conjunction with the existing:

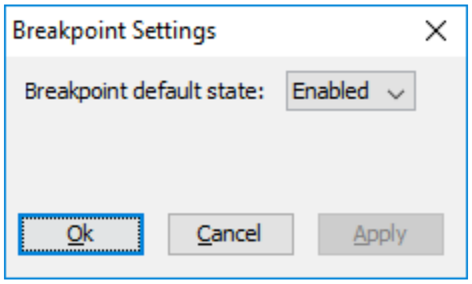
```
iscobol.rundebug=1
```

With the new configuration active, when Remote Debugger starts up and connects to the isCOBOL program's process, the execution can be interrupted or breakpoints can be set using debugger commands such as pause, break, b0 or m0.

The rundebug.auto_pause configuration setting is set to true by default, to maintain the existing behavior.

A new dialog accessible from the "settings" menu has been designed to allow setting the default state of new breakpoints, using a combo-box, as shown in Figure 11, Breakpoint default state. This provides an easy way to set the initial state of newly added breakpoints.

Figure 11. Breakpoint default state

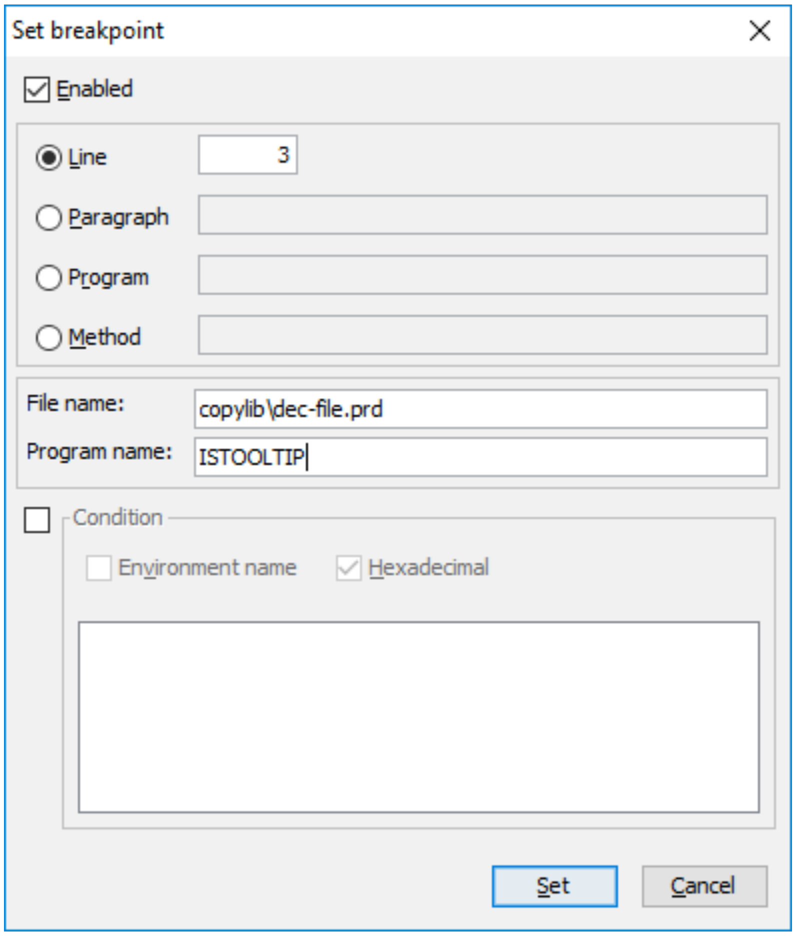


Breakpoint default state: Enabled

The "Set breakpoints" dialog, depicted in Figure 12*, Set breakpoint dialog*, has been improved by allowing you to set a breakpoint for a specific program on a specific file name, typically a copy file. Such a breakpoint will trigger on that file name, at the specified line number, only for a specific program.

If a program name is not specified, the breakpoint will trigger at any program that contains the specific copy file.

Figure 12. Set breakpoint dialog



Set breakpoint

[✓] Enabled

(●) Line: 3
( ) Paragraph
( ) Program
( ) Method

File name: copylib\dec-file.prd
Program name: ISTOOLTIP

[ ] Condition
[ ] Environment name
[✓] Hexadecimal

isCOBOL EIS

isCOBOL EIS, Veryant's solution to write web-enabled COBOL programs, is constantly updated to provide more comprehensive web solutions. In isCOBOL 2019R2 WebClient has received many updates, the HTTPClient class can consume existing web services with attachments, and XMLStream has more powerful xml file handling.

WebClient enhancements

WebClient will now alert the user when an admin or a support user is viewing or recording their session. WebClient's auditing capabilities now log support activities, showing the user name that started viewing, recording, or taking control of the session.

A new option has been added to the application configuration page that allows automatic recording of sessions that are being viewed by authorized support and admin users. With this setting enabled, whenever a user starts viewing a session, it will automatically be recorded.

Session recording can be stopped and resumed multiple times within a support session, allowing finer control of what is being recorded.

HTTPClient improvements

The HTTPClient class is very useful to allow COBOL programs to interact with Web Services. It has been updated to handle Multipart responses by providing several new methods to get the list of received attachments, the list of attribute names for a specific attachment, the values for a specific attribute of a specific attachment, and the content of a specific attachment.

The new methods are shown below:

```
public void getResponseAttachmentIDs(destination)

public void getResponseAttachmentAttrNames(id, destination)

public void getResponseAttachmentAttr(id, attrName, destination)

public void getResponseAttachmentBody(id, destination)
```

XML with qualified tag names

New methods have been implemented in the XMLStream class to better handle XML files when namespaces are involved, making it easier to exchange XML files with third parties.

All the existing methods that write XML files now support an optional Boolean parameter that, when set to true, will generate qualified names in the resulting XML.

This following is the list of methods that support the additional writeQualifiedTagNames parameter:

```
public void write ( Xml-Destination, writeQualifiedTagNames)

public void writeToFile ( Xml-Destination, writeQualifiedTagNames)

public void writeToPrintWriter ( Xml-Destination, writeQualifiedTagNames)

public void writeToStream ( Xml-Destination, writeQualifiedTagNames)

public void writeToStringBuffer (Xml-Destination, writeQualifiedTagNames)
```

The sample code below shows the difference in generated XML file when using the new method with the writeQualifiedTagNames parameter.

Below is the copy file generated by running stream2wrk utility with the command:

```
stream2wrk xml book.xml -p book-
```

```
  >>SOURCE FORMAT FREE
*> XML File: book.xml
*>
*> Generated by isCOBOL-BETA release 2019 R2 build#995.1-20190705-27882
*> Stream2Wrk options: -p book-

01 book-books identified by 'books'
              namespace 'http://somebooksite.com/book_spec'.
   03 book-book identified by 'book'
                namespace 'http://somebooksite.com/book_spec'
                occurs dynamic capacity book-book-count.
      05 book-title identified by 'title' namespace
                        'http://somebooksite.com/book_spec'.
         07 book-title-data pic x any length.
      05 book-author identified by 'author' namespace
                        'http://somebooksite.com/book_spec'.
         07 book-author-data pic x any length.
      05 book-email identified by 'email' namespace
                        'http://somepublishingsite.com/spec'.
         07 book-email-data pic x any length.

    >>SOURCE FORMAT PREVIOUS
```

When invoking the write method without specifying the writeQualifiedTagNames parameter:

```
obj-xml:>write("output1.xml")
```

or using the equivalent new method with writeQualifiedTagNames set to false:

```
obj-xml:>write("output3.xml", false)
```

The XML file created will not contain qualified tag names, as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns="http://somebooksite.com/book_spec">
   <book>
      <title>The Dream Saga</title>
      <author>Matthew Mason</author>
      <email xmlns="http://somepublishingsite.com/spec">
          author@sidharta.com.au
      </email>
   </book>
   <book>
      <title>Sherlock Holmes - I</title>
      <author>Arthur Conan Doyle</author>
      <email xmlns="http://somepublishingsite.com/spec">
          author@GeorgeNewnes.com
      </email>
   </book>
</books>
```

Invoking the write method with the writeQualifiedTagNames parameter set to true:

```
 obj-xml:>write("output2.xml", true)
```

will generated an XML file with qualified tag names, as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:books xmlns:ns1=http://somebooksite.com/book_spec
           xmlns:ns2="http://somepublishingsite.com/spec">
   <ns1:book>
      <ns1:title>The Dream Saga</ns1:title>
      <ns1:author>Matthew Mason</ns1:author>
      <ns2:email>author@sidharta.com.au</ns2:email>
   </ns1:book>
   <ns1:book>
      <ns1:title>Sherlock Holmes - I</ns1:title>
      <ns1:author>Arthur Conan Doyle</ns1:author>
      <ns2:email>author@GeorgeNewnes.com</ns2:email>
   </ns1:book>
</ns1:books>
```