2022

ISSUE 06

TRIANNUAL JOURNAL OF VERYANT AND isCOBOL

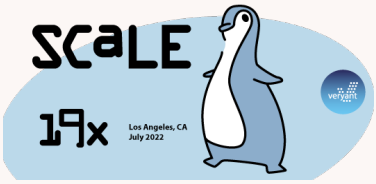
# veryant NEWS

## JOIN US AT SCALE 19X IN LOS ANGELES

### JULY 28-31, 2022

The largest community-run open-source conference in North America is back, and we're setting up our booth as a sponsor.

Stop by and see us for a free gift! More information here




## PLEASE JOIN US ON

Twitter, LinkedIn, or Facebook to up-to-date with Veryant's news

Watch our demonstration videos and Subscribe to our YouTube Channel

## THIS ISSUE



# 2022 R1 and more

**2022R1 has taken isCOBOL a huge step forward in keeping up-to-date and relevent in larger, modern environments**


We now support all the LTS (long term support) releases of Java; Java 1.8, Java 11 and Java 17. You can see the release date and end of support dates for Java versions here The isCOBOL IDE is updated to use the newest stable Eclipse base. 2022R1 is certified to run on Windows 11 with Java 11 or 17. We've added clustering to the WebClient, so now you can balance the WebClient processing load across multiple servers. Look for a video demonstrating how to do this on our YouTube site, here.

2022R1 has a few different ways to expand your configuration file handling. You can piggy-back one file to another with iscobol.conf.copy=<filename>.

You can code to reset all variables to their original values, or set them to values stored in a file, with C$CONFIG. Handling bitmaps and fonts that go with your application is easier now with COPY RESOURCE syntax to include them in your classes.

Avoid hanging background processes by setting iscobol.display.message.timeout so a message box closes automatically. As usual, we have lots of GUI control enhancements, including many for the hamburger menu, adding searching, its own layout manager and several other specializations to this new type of menu that works well on small screens. You can set the main window to darken when you display a modal window that demands the user's attention. These are just some of the new features in 2022R1. You can get more information about all of them in the release documentation and video.

# What are you going to do with your COBOL?

## *An argument for keeping your COBOL rather than starting over, by Daniel Cardenas, Business, Development Director of Latin America*

Recently, many people have talked about COBOL and "ancient" applications, and many outside the COBOL industry don't understand why something wasn't done about the situation before the current crisis.

But those of us who use COBOL every day know it's "easier said than done". Not only does your COBOL application work, it works hard every day. Thinking about changing an essential system with decades of modifications, industry specific applications, with millions of lines of code, is no easy task. Deciding which direction to go is daunting, and I'd dare any decision maker to say they haven't succumbed to "analysis paralysis" on the subject at least once.

The decision makers responsible for deciding what to do with their legacy COBOL programs are at a crossroads. There are two paths to take: keep the COBOL and update it, or throw it away and start over again with a generic package or a total rewrite. Once you start down one path, it's very difficult to change without going all the way back to the crossroads and starting over

My son once cleaned his room by taking everything out of it, then putting everything back again. His theory was that he'd only put back exactly what he needs to, and it would be better organized. And that's basically what happened. However, it took him a whole weekend, so he had to sleep on the couch for two nights during the process and the whole house was a mess during that time. And eventually, his room became just as messy as it had been because things he thought he didn't need keep creeping back into his room over time.

Replacing your COBOL with a generic package or totally rewriting your application is a little like his "slash and burn" method of room cleaning. Yes, you can end up with something bright and shiny. But you'll still have to touch every part of your COBOL application to try to copy its functionality.

And then either pay an outside vendor a lot of money to modify their generic package to fit your business, or wait years for your programmers to write an application from scratch that will replace something it took decades to fine tune.

Updating your existing COBOL comes with problems too, such as who's going to maintain the COBOL if all your COBOL programmers are retiring? And how do you tell your customers that you have a cutting-edge application written in COBOL? How do you pick the right COBOL vendor with the right set of tools to make your update successful?

**"Not only does your COBOL application work, it works hard every day."**

That's where Veryant fits into the decision. Our IDE is based on an industry standard IDE – Eclipse, so most programmers are familiar with the environment already. COBOL is a relatively easy language to learn, and drawing screens and coding in COBOL is a snap when they use the integrated IDE tools we've created.

If you want to minimize your programmer's interactions with COBOL, it's easy to create individual REST services and push your COBOL into a background "black box", and use a more modern language better fitted to graphical displays, like Java, JavaScript, or Python for your user interface code.

isCOBOL compiles your COBOL into Java code, then compiles again into a Java class. As far as your customers or users can tell, you have a Java application.

Perhaps the most important decision of your company may be what to do with your COBOL. I hope you seriously consider keeping it. Call us for a discussion, demonstration, or code analysis.

# isCOBOL IDE Icon

*The IDE uses different icons in the 3 sections of the IDE explorer in order to represent the status of the project. Here's your map to understanding those icons.*

## All sections


isCOBOL project


isCOBOL library: lists the version of isCOBOL library used by the project

## Structural view


isCOBOL screen program


isCOBOL WOW program


Screen designer section for Screen program or Forms list for the WOW program.


Screen designer for Screen program or Form designer for WOW program


Report designer section, containing all the reports of each program.


Report designer


Working storage and Local storage designer


Linkage section designer


File section containing all the datasets (file) of each program.


Dataset (the representation of a file used inside the program)


Event paragraph for Screen program and WOW program

## Icon decorations

Each icon can be decorated with some small symbol to show extra information.

A compiler warning will add a yellow triangle on the low left corner. For example the source icon becomes  and the folder  .

A compiler error will add a red cross on the low left corner. For example the source icon becomes 

and the folder  .

The error icon will override the warning icon.

## File view

 Empty project folder

 Project folder with some files in it

 List folder with some files in it

 Copy file

 List file

 Program source code

 Program id: under each source code there is this icon. This represents the generated class

 Class id: this icon has the same meaning as above, but for a class id

 The container of the methods of a class id

 A method of the class-id

## Data view

 Dataset. The representation of the dataset (file), one for each file in the program

 File description (FD) designer

 File key designer

 File i-o handling designer

 File EFD designer

 File event paragraph designer

## Icon decorations

If the file or the folder are not physically present in the project folder, but they are linked from another location, a link symbol is added in low-right corner of the icon. For example the source icon becomes  and the folder 

These are the isCOBOL decorations. You might see others added by Ecplise plug-ins, like SNV.

## Have You Seen This?

### New YouTube Videos

isCOBOL's Profiler- how it works and a demonstration
isCOBOL's Code Coverage Utility
What's new in 2022R1 – with demonstration
All about isCOBOL Licensing
Customizing isCOBOL
WebClient Clustering

### New Knowledge Base (KB) Articles

How can you wait for several threads to finish?
How to use font-based icons in isCOBOL GUI programs
Can I use COBFILEIO to give access to my c-tree files to a Java program?
Modernize your character application

# Having a Persistent Volume for isCOBOL Server in a Docker

*The second of a three part series on isCOBOL and Docker Containers, this article shows you how to use persistent data in your container. Here's a step-by-step guide from Senior Support Engineer, Valerio Biolchi*

## Prerequisites

The prerequisites on a Linux 64bit computer (where the docker is built) are :

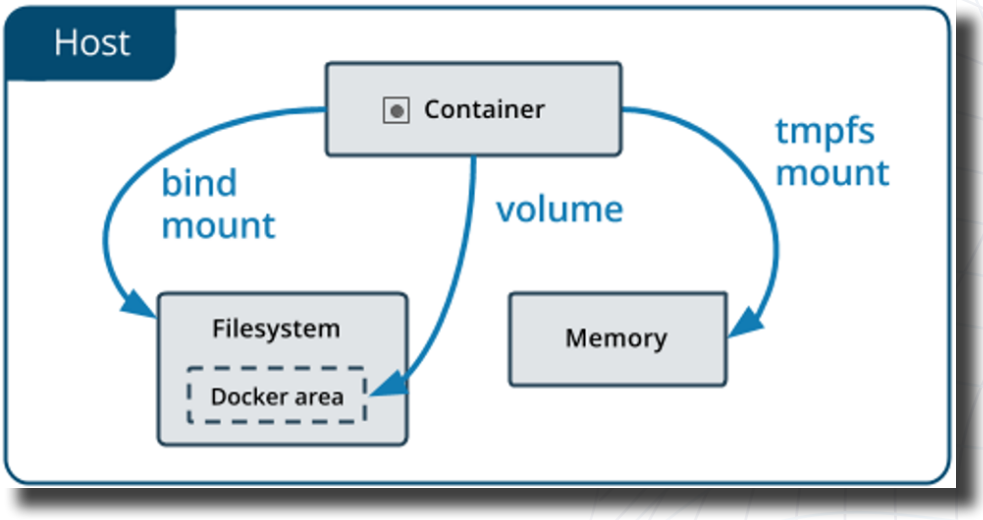
- The Docker engine (https://docs.docker.com/engine/install/) installed from a repository
- A licensed isCOBOL SDK environment
- An isCOBOL Server running in Docker having ISAPPLICATION sample setup and running. See instructions in the first article of the series, here.

## Manage data in a Docker

As default behavior, all files created inside a container are stored on a writable layer. This means that data doesn't persist when the container is removed and it's pretty complex to make data available to another process that run outside container. Also, the container's writable layer is strongly coupled with host machine where container is running and it's not simple to move that data to another host. Last but not least, an additional extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.

Docker system provides two options for containers to store files on the host machine so that the files are persistant even after the container stops: volumes, and bind mounts.

Docker also provides to containers a way to store files in the memory of the host machine called ''tmpfs". Since host machine memory is used, obviously these files are not persistent but are still useful to get the best performance or for security reasons to protect data.



***Volumes*** are stored in the host filesystem where Docker is running. For example, on "/var/lib/docker/volumes/." As a good practice non-Docker processes should avoid modifyng this filesystem. Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.

***Bind mounts*** were the original option for persistent data in a Docker. They can be stored anywhere in the host filesystem. Non-Docker processes on the Docker host or a Docker container can modify them at any time. As a good practice, you should use volumes where possible, however in some cases you might consider to using bind mounts between Docker processes and the Host machine; for example if you need to share a configuration file between the host machine and containers.

## Using Volumes

We can create a volume by using the "create" subcommand and passing a volume "name" as an argument:

```
root@ubuntu# docker volume create myAppVolume
```

The "ls" subcommand shows all the volumes known to Docker:

```
root@ubuntu# docker volume ls
```

```
DRIVER                          VOLUME NAMES
local                           myAppVolume
```

To display detailed information on one or more volumes, we use the "inspect" subcommand:

```
root@ubuntu# docker volume inspect myAppVolume
```

```
[
    {
        "CreatedAt": "2022-04-21T01:18:45-07:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/myAppVolume/_data",
        "Name": "myAppVolume",
        "Options": {},
        "Scope": "local"
    }
]
```

We should note that the driver of the volume describes how the Docker host locates the volume. Volumes can be also defined to be located on remote storage via NFS. In above example, the volume is in local storage.

## Start a Container with a volume

If you start a container with a volume that does not yet exist, Docker creates the volume for you. The following example mounts the volume "myAppVolume" into ":/isapplication/data" in the container

```
docker run --name iscobolserver -v myAppVolume:/isapplication/data -p 10999:10999
-d iscobol
```

The -v option contains three components, separated by colons:

Source directory or volume name
Mount point within the container
(Optional) 'ro' if the mount is to be read-only

Inside the container you should have the ISAPPLICATION setup with a "config.properties" configuration file. The file should include iscobol.file_prefix, the property that defines the directory where COBOL file are created and found, set to the mount point of the defined volume. For example:

```
iscobol.file_prefix=/isapplication/data
```

After Docker container execution, you will find COBOL index data files within the physical folder named "/var/lib/docker/volumes/myAppVolume/_data".

## Docker File

Here's the dockerfile contents used in the previous docker article, for reference:

```
# Dockerfile

FROM  openjdk:11
MAINTAINER  Veryant
ENV ISCOBOL=/var/isCOBOL2022R1
ENV ISCOBOL_CLASSPATH=${ISCOBOL}:${ISCOBOL}/isapplication
ENV ISSERVER_OPTS="-c ${ISCOBOL}/isapplication/config.properties"

RUN mkdir ${ISCOBOL}

COPY iscobol.properties ${ISCOBOL}/iscobol.properties
COPY isCOBOL_SDK2022R1/lib ${ISCOBOL}/lib
COPY isCOBOL_SDK2022R1/bin/isserver ${ISCOBOL}/bin/isserver
COPY isCOBOL_SDK2022R1/sample/isapplication ${ISCOBOL}/isapplication

WORKDIR "${ISCOBOL}/isapplication"

CMD ${ISCOBOL}/bin/isserver run
```

# COBOL/Web Integration

## *The Power of the IWC-PANEL component.*

The IWC-PANEL is a container control that you can use in your COBOL application to include web components in your screen when your program is encapsulated in a web page. The IWC-PANEL control lets you use COBOL MODIFY and INQUIRE verbs to interact with the web component.
In general, an iwc-panel component on the screen section would look like the following:

```
03 f-map iwc-panel
            js-name          "f-map"
            line  5          column  2
            size 68 cells    lines 15 cells
            value            fmap-struct
            event procedure FMAP-PROC.
```

The JS-NAME property holds an identifier that will be sent to the web page upon creation, so that the corresponding web component can be created.
The control's VALUE property holds the message structure used to send actions to the panel in the web page, so you would use the MODIFY statement on that property to send a message to the web page. On the other hand, if the web page executes a performAction on the panel, the event procedure of your COBOL program will be called and an INQUIRE on the value property will return the message that has been sent.

### *TELL THE WEB PAGE TO CREATE THE COMPONENT*

For every IWC-PANEL in a form, a callback in the web page is called, with the details necessary to perform component initialization. The webclientInstance.options .compositingWindowsListener object defines callbacks for various events, such as creating the IWC-PANEL and windows opening and closing.
An IWC-PANEL creation will trigger the windowOpened callback, and a reference to the IWC-PANEL is passed as function argument. The callback can check the .name property to determine which control has been created and react accordingly. Here's a JavaScript code snippet showing the windowOpened handling:

```
compositingWindowsListener:{
   windowOpened: function(win) {
      if (win.name === 'f-map'){
         createMap(win)
      }
   },
},
```

### SEND INFORMATION TO WEB PAGES

The following code snippet shows how the COBOL program interacts with the web page. A message is created using OOP to a jsonStream, then passed to the JavaScript with a simple COBOL "modify f-map value …" statement

```
SHOW-ON-MAP.
    move "selectOffice"        to fmap-action
    move offices(office-index) to selected-office
    set objJsonStream to jsonStream:>
                         new(selected-office, 1);;
    set strbuffer     to string-buffer:>new
    objJsonStream:>writeToStringBuffer(strbuffer)
    move strbuffer:>toString to fmap-data
    modify f-map value fmap-struct.
```

The Javascript receives and parses the data in this snippet:

```
if (actionName === 'selectOffice'){
    let office = JSON.parse(data);
    selectOffice(office);
 }
```

### GET INFORMATION FROM THE WEB PAGE

A change on the map triggers a performAction on the web page which is caught by the COBOL program. The program executes the control's event procedure, FMAP-PROC, which uses an "inquire f-map value …" statement to get the event
The Javascript sent the information in this code snippet:

```
infoWindow.addListener('closeclick', () =>{
   if (mapControl){
      mapControl.performAction(
        {actionName: 'pinClosed',
        data:marker.title});
  }
 });

if (mapControl){
    mapControl.performAction(
    {actionName: 'pinClicked',
     data:marker.title});
}
```

And the COBOL program processed it in the control's event procedure.

```
FMAP-PROC.
     if event-type = ntf-iwc-event
        inquire f-map value in fmap-struct
        evaluate fmap-action
        when "pinClicked"
             move fmap-datato sel-description
        ...
        when "pinClosed"
        ...
        end-evaluate

     end-if.
```

In the sample/issamples folder of your isCOBOL installation we provide a complete project that shows how to integrate a Google map component in a COBOL application, and how to interact with it. Directions for running the SAMPLES program encapsulated in a web page – a requirement to use the IWC-PANEL, is in the README.md document found in the issamples folder.

## Documentation Highlights

### Common JDBC connection settings

You can manage your data in every JDBC-compliant database with isCOBOL.

We offer two ways for you to do this. It can be done by writing embedded SQL (ESQL) code in your program or you can use our DatabaseBridge product to write the ESQL for you.

In both cases, you 'll need to add the proper JDBC driver library to the Classpath and configure the JDBC connection string.

To make it easier for you to do that, the isCOBOL documentation includes a collection of JDBC drivers and connection strings for the most common relational databases here.

## Customizing Default Icons

The isCOBOL Framework includes a series of default icons that are used in various places of the GUI. These icons are PNG and GIF files stored in the com.iscobol.gui.client.swing package in the iscobol.jar library. It's possible to customize these icons by adding a library (or a folder) with the same package before iscobol.jar in the CLASSPATH. Suppose that you wish to customize the funnel icon shown on the Grid's heading when either the FILTERABLE-COLUMN style or the FILTER-TYPES property is set.

# Symbol Font Advantages

***Using the WBITMAP-LOAD-SYSTEM-FONT and WBITMAP-LOAD-SYSTEM-FONT-EX opcodes***

Symbol fonts are font libraries. Here are some of the advantages of using symbol fonts for your images rather than external, separate, image files.
Traditional icons lose quality when you increase or decrease their size, or open and save them in a photo editing program. Icon fonts used by these W$BITMAP opcodes store images in a vectorial format and don't lose quality. This means that with symbol fonts:

- If you need more than one size of an icon (ex: 32 and 16 pixels), you don't have to create and maintain two separate icon files to keep the same quality.
- When you add an icon to the end of a strip you keep the quality of the strip by adding code, rather than having to open the strip in a photo editing program, degrading quality.
- If your application has two different schemas (ex: light and dark) you can set these properties in your code, instead of creating two sets of images with different colors.
- Symbol fonts offer a huge variety of images. For example Font Awesome, one of the most popular system fonts, has over 7,000 images.

Here's an example of code that changes the size and color scheme of a defined strip:

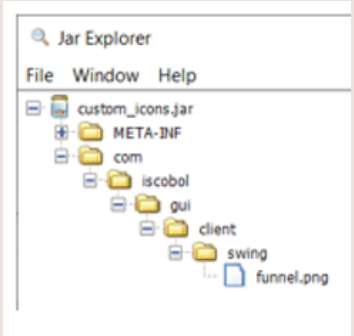
```
If use-big-image
   move 32 to imageWidth
else
   move 16 to imageWidth
end-if

If dark-theme
   move -13421772 to imageColor
else
   move -6710886  to imageColor
end-if

CALL "W$BITMAP" USING
      WBITMAP-LOAD-SYMBOL-FONT
                  fontHandle
                  charactersSequence
                  imageWidth
                  imageColor
            GIVING bitmapHandle.
```

You can read more about how system fonts work in our KB article here.

You just have to place a file named funnel.png in a folder structure named com/iscobol/gui/client/swing and then create a jar from it, e.g. Copy this jar to the "jars" directory of your isCOBOL SDK and, from now on, when you run a COBOL program using the isCOBOL SDK, you will see your custom funnel on grid headings. Refer to the online documentation here for the list of icons that you can customize and for more details about the creation of your custom jar.



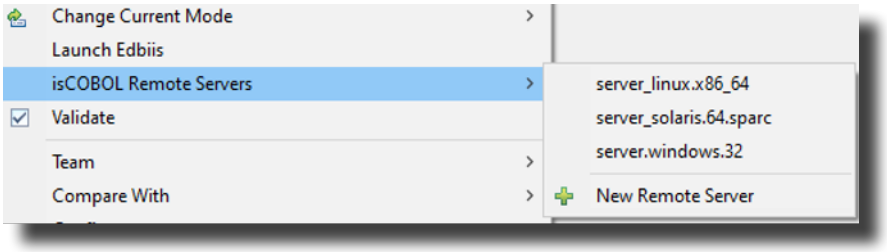
# Use the IDE to run your remote batch programs

Your IDE can compile and execute your code locally or remotely, with the IDE's remote server. If you have batch processes that you need to run on a remote server, with a little setup you can launch them from the IDE instead of directly on the server.

Once you start isCOBOL's Application Server with the special -ide switch on the remote server, you create a project and define the remote server in your IDE's workspace, then link, or bind them together. Once you've done that, you have more compile and runtime modes – in addition to the normal "run" you will find "@<your server name>.Run" for instance.

You can use the remote server to run more than batch programs. Programs with a user interface run well too, because the remote server uses thin client technology.

Details are here in our documentation, and support is just an email away!




# Where are my licenses!?!

The runtime framework (and other isCOBOL products) looks for licenses in a file called "iscobol.properties" in 4 places, and a 5th place and name you specify. The most recent license found is used. Here's the list of locations and order searched.

To see what licenses the runtime sees, run

```
iscrun -license
```

You'll get a list of all the "iscobol.properties" files found and what licenses were picked up and in what order.

1. `/etc/iscobol.properties`
2. `<userhome>/iscobol.properties`
3. `<java classpath>/iscobol.properties`
4. `-c <any path and name passed from the command line>`
5. `$ISCOBOL/iscobol.properties`

veryant

# UNDECORATED WINDOWS

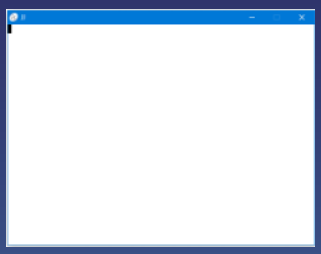
*What they are and when it makes sense to use them.*

When the undecorated style is set, native decorations like the frame and title bar are not shown.

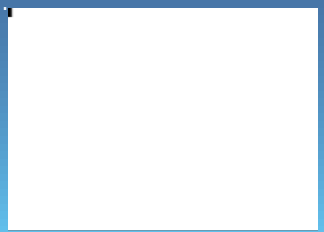
This windows style is useful in the WebClient environment where you might want to remove the title bar of the window so the program screen will look like a common HTML form or is embedded in a page.

You can use this style in floating, independent, and initial (standard) graphical windows. It has no effect on Docking and MDI windows, and Notification windows are already undecorated by default.

Here's a decorated window:



And here's the same window with the "undecorated" property. Notice the title bar and the blue edging (frame) are gone.



Evolution, without revolution

## Contact Us

For supported customer email us at **support@veryant.com**

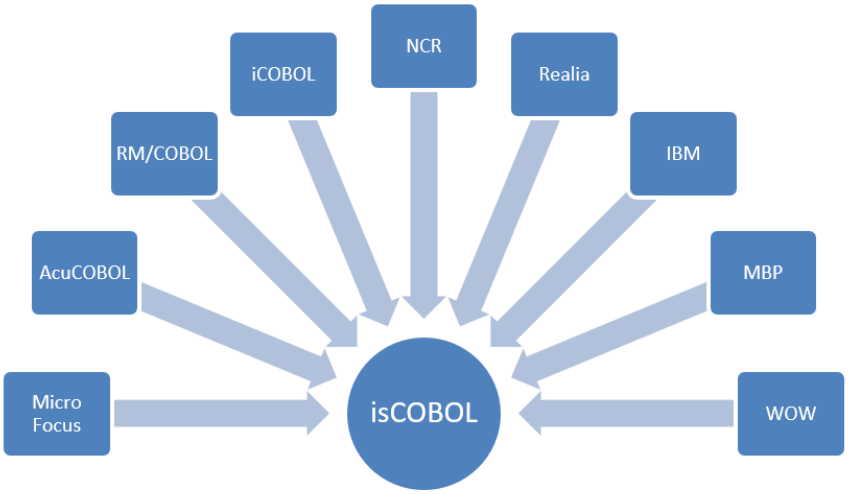
If you would like Veryant to contact you to schedule a technical product briefing, email us at **info@veryant.com**

If you would like Veryant to contact you for special quote or sales assistants email us at **sales@veryant.com**


Corporate Headquarters
6390 Greenwich Dr., Suite 225
San Diego, CA 92122 - USA
Tel (English): +1 619 797 1323
Tel (Español): +1 619 453 0914
info@veryant.com

European Headquarters
Via Pirandello, 29
29121 - Piacenza - Italy
Tel: +39 0523 490770
Fax: +39 0523 480784
emea@veryant.com


**As always, 2022R1 contains multiple compatibility additions – as we continue to make your conversion process as smooth, quick, and pain-free as possible.**




veryant.com

Follow **Veryant** on

**veryant.com**