2021

ISSUE 05

TRIANNUAL JOURNAL OF VERYANT AND isCOBOL

veryant NEWS

VERYANT'S OFFICE IN PIACENZA, ITALY

Be sure to watch the video of scenes from the 2021 Distributor's conference.

Even if you weren't there, you'll be interested in seeing the Piacenza office and many of the Veryant people you may recognize or talk to regularly.

The office's lobby has several unique features, including a car hanging from the wall. See the "Have you seen this" section on page 8.

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

2021 R2 is here

Veryant is pleased to announce the latest release of isCOBOL™ Evolve, isCOBOL Evolve 2021 Release 2.

The biggest addition in 2021 R2 is the ability to embed, or encapsulate, a COBOL application within a web page **without modifying the COBOL**! By enabling your isCOBOL to talk to webpage languages with the new IWC routines and GUI control, we've made it easy to keep your COBOL and work with your webpage developers to make an integrated view of two previously separate environments.

- You no longer need to include the location of the /isdef folder for the compiler to find isCOBOL's definition files. It's done autmatically.
- Linkage variables can now be fixed OR variable length, the runtime will handle the parameter as it is passed without an error or lost data.
- In the Application Server, you could reload any program found in the code_ prefix paths. This is useful when you modify a program and want to force it to be loaded in place of the old class by unloading the old class from the Application Server's memory. In 2021R2 you can unload ALL the classes and force the Application Server to start with a fresh copy of all the classes.
- We've increased the performance of printing significantly, both by performing the print in a separate thread, and by using improved TCP packet handling.
- 2021R2 also has improvements in Encoding in the HTTPClient EIS class and handling Json in the HTTPClient and HTTPHandler classes.

For more information about the new 2021R2 changes you can go to our website by clicking here.

Installing the isCOBOL Server using Docker

Starting the Application Server in a docker container is easy and efficient. Here's a step-by-step guide from Senior Support Engineer, Valerio Biolchi

In order to simplify the process to use isCOBOL Server in a docker container we've created a guide to show you how to build, install and start a Docker repository to run an isCOBOL Application Server on a Linux 64bit machine.

Docker is a way to provide drop-in solutions to deliver software in packages called containers, similar to what Platform-as-a-service (PAAS) would offer. Containers run in an isolated environment and contain their own software, libraries and configuration files. The containers can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines.

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
- Your COBOL application running in Thin Client mode

Instruction File

We will start with an instruction file named "Dockerfile". Because isCOBOL needs Java to be executed, the isCOBOL repository will be based on the OpenJDK image. All docker command references here can be found at https://docs.docker.com/engine/reference/builder.

```
# Dockerfile
FROM. openjdk:11
MAINTAINER  Veryant
```

Then we can set environment variables to be used during the build and from the commands invoked by the docker image with ENV.

```
ENV ISCOBOL=/var/isCOBOL2021R2
ENV ISCOBOL_CLASSPATH=${ISCOBOL}:${ISCOBOL}/iscontrolset
```

The RUN command creates the isCOBOL main folder within the docker container. RUN executes commands in a new layer and creates a new image. E.g., it is often used for installing software packages.

```
RUN mkdir ${ISCOBOL}
```

Now in order to start our Cobol application thru the isCOBOL Application Server, we need to copy the appropriate files to the docker container. COPY copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>:

```
COPY yourlicense.properties ${ISCOBOL}/iscobol.properties
COPY isCOBOL_SDK2021R2/lib ${ISCOBOL}/lib
COPY isCOBOL_SDK2021R2/bin/isserver ${ISCOBOL}/bin/isserver
COPY isCOBOL_SDK2021R2/sample/iscontrolset ${ISCOBOL}/iscontrolset
```

Finally, we set the command that will run the isCOBOL Application Server when the docker image is started. CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.

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

Here are the full contents of the Dockerfile we've created to have a container fully working with isCOBOL Server:

```
# Dockerfile
FROM  openjdk:11
MAINTAINER  Veryant
ENV ISCOBOL=/var/isCOBOL2022R1
ENV ISCOBOL_CLASSPATH=${ISCOBOL}:${ISCOBOL}/iscontrolset
RUN mkdir ${ISCOBOL}
COPY yourlicense.properties ${ISCOBOL}/iscobol.properties
COPY isCOBOL_SDK2022R1/lib ${ISCOBOL}/lib
COPY isCOBOL_SDK2022R1/bin/isserver ${ISCOBOL}/bin/isserver
COPY isCOBOL_SDK2022R1/sample/iscontrolset ${ISCOBOL}/iscontrolset
CMD ${ISCOBOL}/bin/isserver run
```

The "CMD" line above uses isserver's 'run' command, which is new in our next version, 2022R1. To use the current or older version, you can start isserver with a script file like run.

sh below:

```
#!/bin/sh
java -cp .:${ISCOBOL}/lib/*:${ISCOBOL}/iscontrolset com.iscobol.as.AppServerImpl
wait $!
```

Then replace the CMD line in the Dockerfile above with these lines to copy and run the script:

```
COPY run.sh ${ISCOBOL}/bin/run.sh
CMD ${ISCOBOL}/bin/run.sh
```

Build the image

Next, we can build the "iscobol" docker image from the folder containing the Dockerfile and your license. Note that the repository name (iscobol in this example) must be lowercase.

```
root@ubuntu:/home/myDocker/myApp# ls
Dockerfile isCOBOL_SDK2021R2 yourlicense.properties
root@ubuntu:/home/myDocker/myApp# docker build -t iscobol .
```

Now we can see the iscobol image by listing all the docker images, showing their repository name and other information:

```
root@ubuntu:/home/myDocker/myApp# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
iscobol	latest	8e66cb700e21	About a minute ago	706MB
postgres	latest	317a302c7480	4 weeks ago	374MB
mysql	5.6	f3b364958c23	6 weeks ago	303MB
ibmcom/db2	latest	ced2e4b31b7a	8 weeks ago	2.97GB
mcr.microsoft.com/mssql/server	2019-latest	80bdc8efc889	8 weeks ago	1.55GB
store/oracle/database-enterprise	2019-latest	12a359cd0528	4 years ago	3.44GB

Create the Container

Now we want to create a writable container named "**iscobolserver**" based on the "iscobol" image. We do this by running the command specified as the CMD tag in the Dockerfile.

At times, you may need to set out some networking rules to enable smooth interaction between containers in multi-container applications or make your Docker ports accessible by services in the outside world. An option is to use -p flag or -P flag in the Docker run string to publish the internal isCOBOL Server default port 10999 to external 10999 Here's a command to run the docker, specifying the name of the container, matching the internal and external ports and specifying the image to use to create the container:

```
docker run --name iscobolserver -p 10999:10999 -d iscobol
```

Now we can check the status of our “iscobolserver” container.

```
root@ubuntu:/home/myDocker/myApp# docker ps -a
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
6e8c9e89ffb3   iscobol   “/bin/sh -c ‘${ISCOB…”   3 seconds ago   Up 2 seconds

                                PORTS                                                 NAMES
                                0.0.0.0:10999->10999/tcp, :::10999->10999/tcp         iscobolserver
```

It shows that the isCOBOL Application Server is listening on port 10999 of the computer where the docker is running it and all isCOBOL thin client users can use that service.

Stopping the service

We can stop the container and terminate the execution of the isCOBOL Application Server with this command:

```
root@ubuntu:/home/myDocker/myApp# docker stop iscobolserver
```

Checking the docker process again we can see its status is “Exited”:

```
root@ubuntu:/home/myDocker/myApp# docker ps -a
```

```
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
6e8c9e89ffb3   iscobol   “/bin/sh -c ‘${ISCOB…”   28 minutes ago   Exited (137) About a minute ago

                                                                  PORTS          NAMES
                                                                                 iscobolserver
```

What’s Next?

Look for our article in the next newsletter where we will discuss how to create Docker Volumes as a mechanism for your persistent isCOBOL data in a container or shared between containers.

Documentation Highlights

A history of Veryant's software

The Release Overview book in Veryant's documentation set includes highlights of the new features introduced by the current version, but it also includes the highlights on features introduced by previous versions.

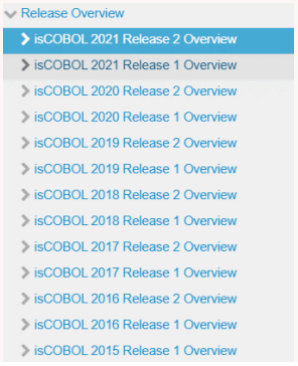
Did you miss a couple of releases and want to know what changed before upgrading to the current one?

Do you want to track how a specific product evolved from version to version?

Are you interested in knowing when a specific product was introduced?

The Release Overview book is the best place for this kind of reading. The collection of Release Overview documents in the documentation ranges from 2015 to now:

6 years of history!



Faster Printing in 2021R2

Performance of print jobs have been improved in 2021R2, both as a result of the async mode implementation thru the new configurations:

iscobol.print.spooler_async=true|false (default: true) to set the print job to be run asynchronously
iscobol.print.pdf_async=true|false (default: false) to have the PDF print job to be executed asynchronously.

and with better Application Server TCP packet handling when running in ThinClient. This chart shows the gains of printing using the new 2021R2 release compared to the previous 2021R1.

Standalone tests	2021R1	2021R2
character print job with 200 pages on -P SPOOLER	18,27	6,35
graphical print job with 20 pages on -P SPOOLER	11,37	4,38
ThinClient tests	**2021R1**	**2021R2**
character print job with 200 pages on -P SPOOLER	16,93	1,75
graphical print job with 20 pages on -P SPOOLER	69,80	14,75
character print job with 200 pages on -P PDF	17,53	13,70
graphical print job with 20 pages on -P PDF	64,86	22,05
character print job with 200 pages on -P PREVIEW	6,79	2,04
graphical print job with 20 pages on -P PREVIEW	59,10	14,25

All times are in seconds. Hardware details of client machine: Windows 10 Pro i7-8550U CPU @ 1.80GHz 16GB
Hardware details of server machine: macOS Big Sur Apple M1 16GB.

HAVE YOU SEEN THIS?

NEW YOUTUBE VIDEOS

2021R1 New Features

WebClient 2021R1 Demo

DatabaseBridge (from the archives)

Veryant's Distributor's Meeting 2021

2021R2 New Features

Veryant's Load Balancer

NEW KNOWLEDGE BASE (KB) ARTICLES:

Can I define one or more data items based on the definition of another one?

Modernizing your COBOL application by using isCOBOL Compiler code injection

Why start using the OCCURS DYNAMIC? How to save memory by replacing your fixed arrays

A giant leap forward for WEB enablement

We have exciting news about the new capability to mix HTML/JavaScript development with your isCOBOL desktop applications through WebClient. This is useful

- **to develop customized web pages with areas where isCOBOL applications can run and interact with the encapsulating web page**
- **to develop isCOBOL desktop applications with custom web components embedded in the application screen.**

When encapsulateing your application in a web page, everything on the web side is done with JavaScript using a container web page. The client side uses isCOBOL's new IWC$ library routines.

The isCOBOL WebClient now allows the interchange of JavaScript messages between the isCOBOL desktop application and the underlaying web page, Your code will start and stop the messaging engine using the new library routines *IWC$INIT* and *IWC$STOP*. The isCOBOL app sends a message to the web page using the *IWC$SEND* routine. When the web page sends a message to the isCOBOL app, it is received using the *IWC$GET* routine.

Here's an example of how your would use those 4 routines:

```
78  78-iwc-crt-status           value 1001.
77  data-to-send                pic x any length.
01  iwc-struct.
    03 iwc-action               pic x any length.
    03 iwc-data                 pic x any length.
    03 iwc-bytes                pic x any length.

ACTIVATE.
    call "IWC$INIT" using 78-iwc-crt-status
SEND-TO-HTML.
    initialize iwc-struct.
    move "ComSample"  to iwc-action
    move data-to-send to iwc-data
    call "IWC$SEND"   using iwc-struct
READ-DATA-FROM-HTML.
    initialize iwc-action
    call "IWC$GET" using iwc-struct
    if iwc-action = "EXECUTE_PGM"
       call IWC-DATA
    end-if.
DEACTIVATE.
    call "IWC$STOP" giving return-code.
```

You can find an in-depth example in the $ISCOBOL/samples/webclient/encapsulated/source folder, which uses entry-points in IWC.cbl to respond to messages. Another sample is in $ISCOBOL/samples/issamples/s-routines/IWC.

If, on the other hand, you want to include a web component written inHTML/JavaScript on your isCOBOL application screen when run in the WebClient, you would use the new IWC-PANEL GUI control. Here's an example of that control's screen section description:

```
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.
```

This control is easy to use for both the COBOL programmer and the JavaScript programmer, because there's no language crossover – the programmers don't have to use an unfamiliar language. If the COBOL programmer wants to send a message from the IWC-PANEL GUI control, they would use the MODIFY statement, and if they want to receive a message, they would use the INQUIRE statement.

A sample program using the IWC-PANEL is in the $ISCOBOL/samples/issamples/s-gui/IWC-PANEL.cbl program.

To learn more about these capabilities, you can

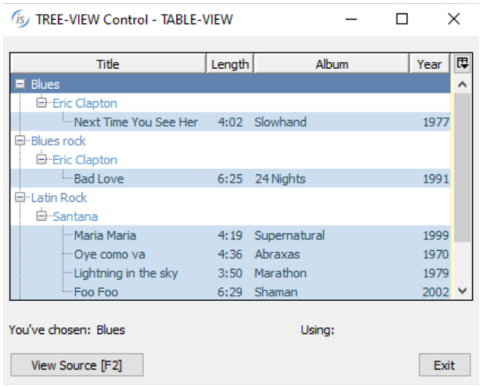
- look at the samples already mentioned
- read the documentation here.
- Watch a video demonstrating encapsulated WebClient here.
- Watch a demonstration of the new features in the 2021R2 New Features video here.

An New Style Updates and Expands your Tree-View Controls

When this style is set, items can have multiple values instead of a single value. Each value is displayed in a separate column. Columns are defined by the Display-Columns property.

Example - Define a tree table view with 3 columns:

```
screen section.
  ...
  03 screen-1-tv-1 Tree-View
     line 2.7
     column 3.4
     size 20.8 cells
     lines 29.1 cells
     height-in-cells
     color 144
     id 2
     table-view
     display-columns
          (1, 10, 15)
```



How to change the Default Printer Font

Simple print programs that produce a text-only output without calling any specific printing library routine don't set a specific font for the print job.

The font is left undefined so it's printer duty to choose which font should be used.

This might produce different output when printing on different printers as well as undesired output when the chosen font is not a fixed pitch.

In order to avoid this kind of issue, you can set the *iscobol.print.default_font* configuration property.

With this property you specify the font that should be used by default when the program doesn't force any font via printing library routines.

For example, in order to have Consolas size 12 as default, set:

iscobol.print.default_font= Consolas-12

veryant

TEMPORARY FILES IN MEMORY

Take control of your temporary file access performance by storing your temporary data in memory instead of on disk.

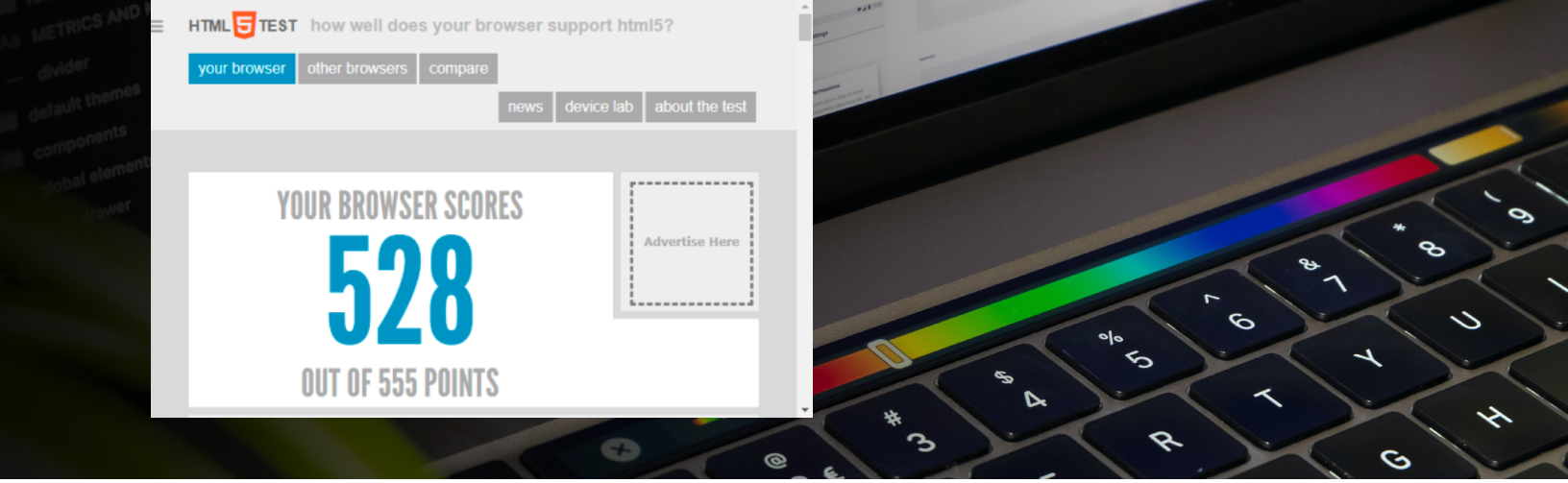
It is a regular practice in COBOL applications to use temporary sequential files. In some cases accessing these files could negatively affect the performance of the applications because the runtime needs to go back and forth to access these files where they are stored on disk.

Fortunately, there is a method to create and access those temporary sequential files in memory, which improves your programs' performance. This solution is very easy to implement: you just need to define the file inside the program as follows:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
select my-file
      assign to address
           memory-area
....
WORKING-STORAGE SECTION.
77 memory-area
      pic x any length.
```

The "memory-area" variable will contain all the records inserted in the "my-file" sequential file, making it faster to access.

For more information and a sample program, see the KB article here.



NEW CHROMIUM WEB-BROWSER

We've added a third web-browser implementation to accommodate complex Javascript and CSS syntax

The Web-Browser control has always given you a choice of two different implementations: the **DJBrowser** component based on SWT or the **JavaFX Webview** component. You could chose which implementation by setting the iscobol.gui.webbrowser.
class configuration property.
However neither of these two implementations is fully comparable to a real web-browser like Firefox, Chrome or Edge.These two web-browser implementations are good for rendering most html pages, but pages with modern and complex Javascript or CSS syntax might not work correctly.

These two web-browser implementations are good for rendering most html pages, but pages with modern and complex Javascript or CSS syntax might not work correctly.
Therefore, we've added a third web-browser implementation that you can activate via the *iscobol.gui.webbrowser.class* configuration property: **JxBrowser**. This is a powerful web-browser based on Chromium, able to render all the html content the same way as Chrome and Edge browsers.
The JxBrowser can be downloaded from the TeamDev website. See here in the isCOBOL documentation for more details.

Why Start Using the OCCURS DYNAMIC Syntax

Every COBOL application has many programs containing definitions with OCCURS in Working Storage or Linkage sections. These arrays are usually of a fixed size to accommodate the maximum number of references possible in the table. The result is that the memory consumed for the array is fixed, even when the program doesn't need to use all the occurrences. For example, this definition:

```
78  max-element     value 900.
01  w-group.
   03 w-element     occurs max-element.
        05 w-cod    pic 9(3).
        05 w-desc   pic x(20).
        05 w-note   pic x(100).
```

will allocate the memory for all 900 occurrences at the program startup, even if the program will use only few references.

To reduce the memory consumed by allocating only the memory really necessary, you can change your code to use a dynamic occurs. For example:

```
01  w-group.
   03 w-element     occurs dynamic capacity max-element.
        05 w-cod    pic 9(3).
        05 w-desc   pic x(20).
        05 w-note   pic x(100).
```

For more information and sample programs demonstrating how the dynamic arrays work, see the KB article here.

Evolution, without revolution

Contact Us

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

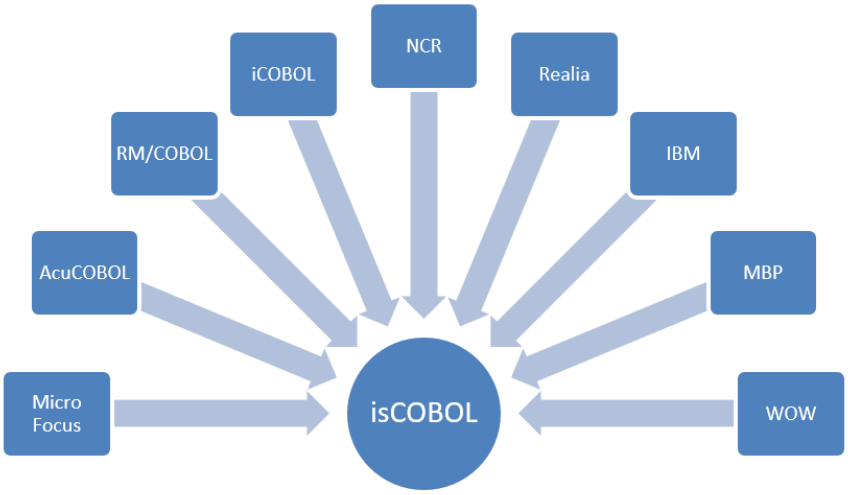
If you would like Veryant to contact you to schedule a technical product briefing, email us at **info@veryant.com**

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

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

As always, 2021R2 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