2025

ISSUE 13

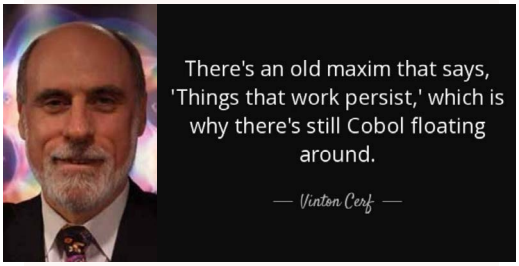
QUARTERLY JOURNAL OF VERYANT AND isCOBOL

veryant NEWS

THIS ISSUE

This newsletter is chock-full of helpful information, including:

- The third and last part of a 3-part series on the debugger as well as a list of common debugging techniques,
- Tips on significantly improving performance of your DatabaseBridge RDBMS table access, and
- A peek at what's coming in a future version.



2025R1 – Barcodes and more!

We've made barcode and QR code generation easier with isCOBOL Evolve 2025R1.

The newest isCOBOL Evolve adds a new GUI Control, barcode and QR-code generation, improved debugging and more.

The new split-pane control gives the user control to resize left and right (or top and bottom) panels. It's easy now to generate 16 different styles of barcodes and QR codes to send to a third party, print out, or display on your screen.

And we've started our revamp of our graphical debugger with improved navigation and color-coded error messages.

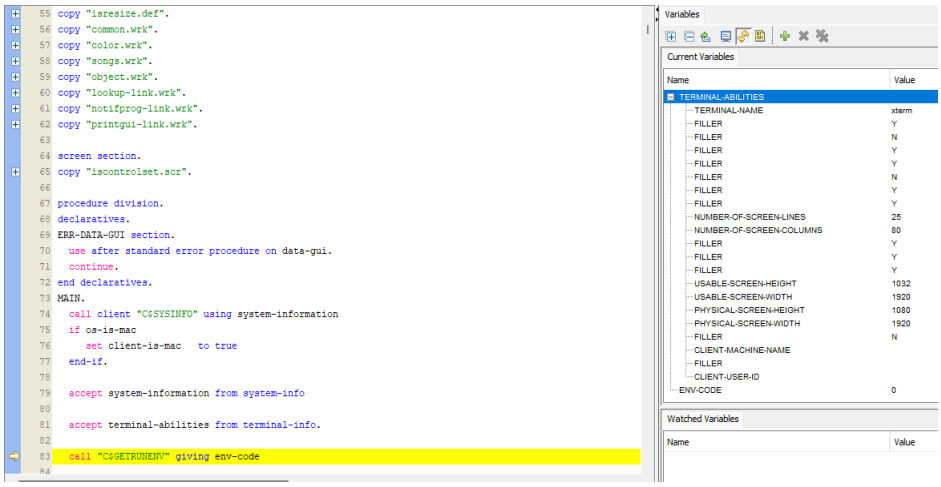
isCOBOL Evolve has had a DCI dll interface to DBMAKER for many years, but with 2025R2 we've added this popular database to our stable of databases accessible with the Database Bridge, giving you a 100% Java access solution.

Are you running an older version of isCOBOL? We're here to help you upgrade to stay with a supported versions and take advantage of new technology. Check out our end-of-life policy here.

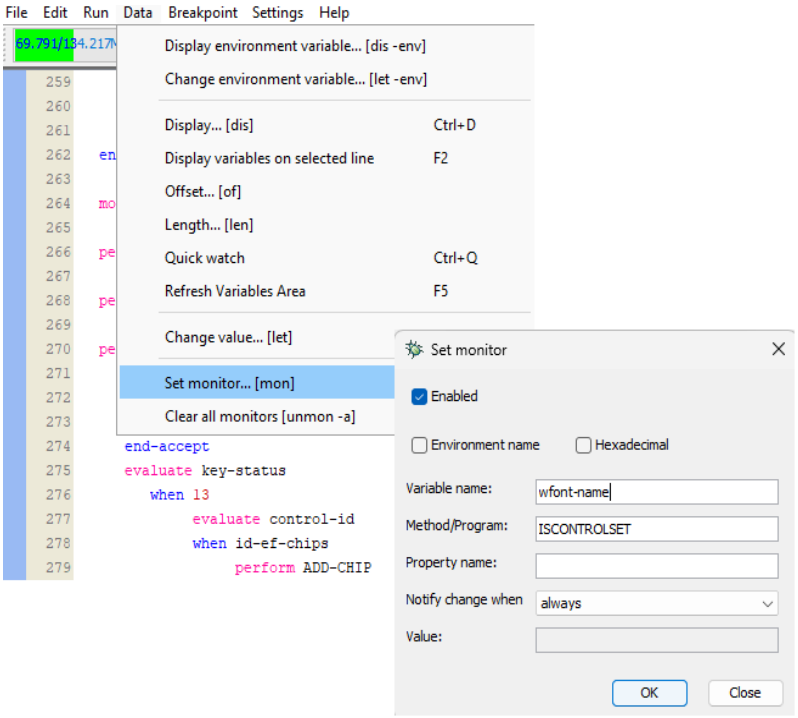
Working in the Debugger

The final installation of a three part deep-dive into the isCOBOL GUI debugger will cover some of the advanced functionality that will help you easily understand your program, how it runs and what it's doing..

The debugger is packed with powerful features designed to give you real-time insights as you step through your code. One standout feature is the **Current Variables** panel, which displays the variables involved in the current line of execution, followed by those from the previous line—making it easy to track changes and understand program flow.



You can set a conditional monitor to stop when a variable changes it's value, or just when it matches some value or values.

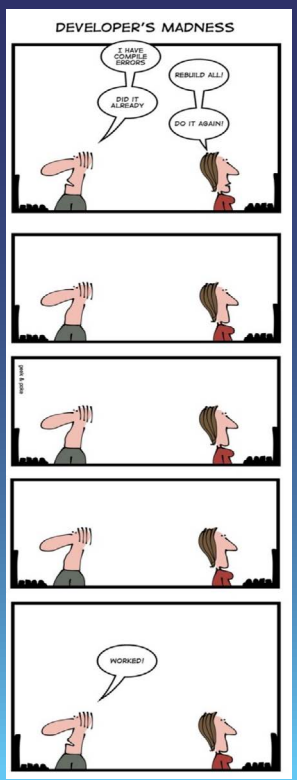


TECHNOLOGY PREVIEW

Veryant will soon launch its **NoSQL Bridge**, a lightweight HTTP/HTTPS interface for direct access to ISAM data,no SQL layer required. This tool allows you to expose ISAM files to web applications while preserving their native format, eliminating the need for intermediate programmatic data transformation.

By decoupling the data layer from the web server and accessing it via Veryant's secure file server, the architecture enhances data security. The RESTful API can be deployed using the built-in Jetty server or externally via a .war file on servlet containers like Apache Tomcat. JSON responses are automatically structured based on the field definitions in the EFD file (<filename>.xml), generated using the -efd compiler option.

For more information contact your sales representative at sales@veryant.com.



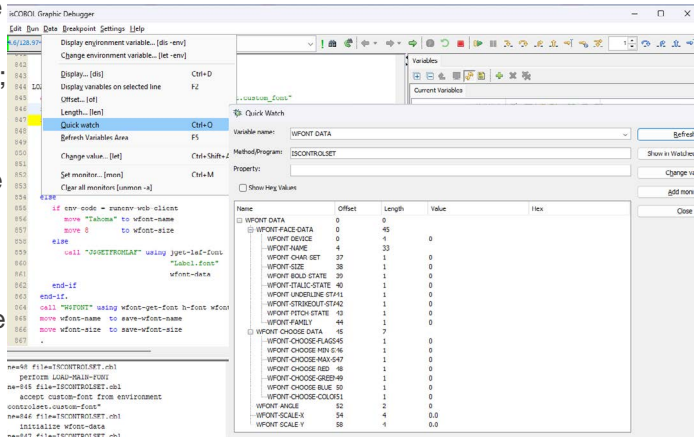
PLEASE JOIN US ON

Twitter, LinkedIn, or

Facebook for up-to-date with Veryant's news

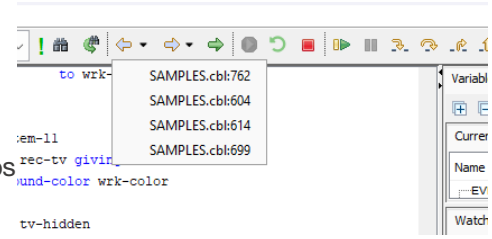
Watch our demonstration videos and Subscribe to our YouTube Channel

Opening your variable in Quick Watch gives you even more power; you can add it to the Watched Variables area, modify the value on the fly, or add it as a monitor. The Quick watch area displays your variable structure in a tree layout automatically.



While navigating your code in the debugger, it's common to jump between different sections—examining paragraphs, variable structures, and more. This can make it easy to lose track of where you are or where the current execution line is.

To help with this, the debugger includes three navigation arrows: Backward, Forward, and Current Line. These tools let you quickly return to previous locations, move forward again, or jump straight to the current line of execution. The debugger keeps a history of your navigation, so you can retrace your steps with ease..



Have You Seen This?

Newest Video:

It's important to keep up with new features and changes in isCOBOL Evolve, even if you don't intend to migrate to that version. This video discusses the new 2024R2 features and additions to isCOBOL products.

And this one covers the newest 2025R1 release enhancements. Both include a demonstration of some of the features.

New KB Articles:

How to improve the performance of your application by making your Tab-Control a container.

How to identify and kill "orphan" Application Server connections.

How to read records faster with isCOBOL Database Bridge

The isCOBOL Database Bridge helps you access relational databases while using standard COBOL I-O statements, with no or minimal changes to the existing code that you use to work on ISAM files. However, there can be a cost in performance when accessing a large number of records, especially on the reading of records using START and READ statements. In this article we'll talk about how these statements are simulated on the relational database and what we can do to optimize them.

Let's use an example with an indexed file containing a primary key of 3 segments and a flag field that is not part of any key;

```
01 file-record.
   03 file-key1.
      05 fk1-cod  pic 9(9).
      05 fk1-year pic 9(4).
      05 fk1-prog pic 9(5).
  ...
  03 file-flag   pic x(1).
  ...
```

In our example database, the file contains over 1,000,000 records, but only 100 of them have fk1-cod = 120 and fk1-year = 2021, and only 10 records in these 100 have file-flag = "Y". Our goal is to extract these 10 records. The COBOL logic to to do that might look like the following:

```
move 120  to fk1-cod.
move 2021 to fk1-year.
move 0    to fk1-prog.
start the-file key >= file-key1.
perform until exit
   read the-file next
      at end
         exit perform
      not at end
         if fk1-cod not = 120 or fk1-year not = 2021
            exit perform
         end-if
         if file1-flag = "Y"
         *> you've found one of the records you were looking for
         end-if
   end-read
end-perform.
```

Because there are three segments in the key, COBOL rules say that the above START statement will read the file using these three steps in order to get a good start:

- all the records that have fk1-cod=120 and fk1-year = 2021 and fk1-prog > 0
- all the records that have fk1-cod=120 and fk1-year > 2021
- all the records that have fk1-cod > 120

The DatabaseBridge needs to mimic these three steps by performing three separate SELECT queries and generate three result sets that could potentially contain thousands of records each. The more key segments, the more queries and result sets, with each query extracting more record than the previous one. READ NEXT operations fetch these result sets. When a resultset has been processed, the next one is processed. When all resultsets have been processed, the end-of-file condition is raised. The same logic applies to READ PREVIOUS operations

.

Easydb-limit_dropdown configuration

In our example COBOL code, the IF statement that exits from the PERFORM cycle when the year is no longer equal to 2021 will avoid the execution of the third query, but the second query will still be performed and we don't need it. Only the first query should be performed for our purposes.

To instruct the Database Bridge to use only the first query (the one that checks all the key segments), you can set the *easydb.limit_dropdown* configuration to 3 before the START statement.

```
set environment "easydb.limit_dropdown" to 3.
```

Note that this setting will be enabled from now on, so in order to avoid affecting future START statements it's good practice to remove it after the START has been performed.

```
call "C$UNSET" using "easydb.limit_dropdown".
```

Edbi-where-constraint

After reducing the number of queries to one we can improve this query by adding additional filters. We know that our program is looking only for those records that have file1-flag = "Y", but the code asks for all the records and filters them in the COBOL program. It would be faster to have the database do this filtering and return only the valid records.

You can add this filtering criteria to a WHERE clause of the DatabaseBridge query with the *edbi-where-constraint* external data item. In our example we will add "FILE1_FLAG= 'Y'" to the where constraint data item.

```
move "FILE1_FLAG = 'Y'" to edbi-where-constraint.
```

Note that like the dropdown limit, this setting will be enabled from now on so in order to avoid affecting future START statements it's good practice to clear the external data item afer the START has been performed.

```
initialize edbi-where-constraint.
```

These changes results in 6 additional statements in your code, with no refactoring, and at runtime the new code will perform much faster with the DatabaseBridge. Here's what your new code would look like:

```
 WORKING-STORAGE SECTION.
 . . .
     exec sql include sqlca end-exec.
 01   edbi-where-constraint  pic x(300) is external.
 . . .
move 120  to fk1-cod.
move 2021 to fk1-year.
move 0    to fk1-prog.
move "FILE1_FLAG = 'Y'" to edbi-where-constraint.
set environment "easydb.limit_dropdown" to 3.
start the-file key >= file-key1.
perform until exit
   read the-file next
      at end
         exit perform
      not at end
         if fk1-cod not = 120 or fk1-year not = 2021
            exit perform
         end-if
         if file1-flag = "Y"
         *> you've found one of the records you were looking for
         end-if
   end-read
end-perform.
initialize edbi-where-constraint.
call "C$UNSET" using "easydb.limit_dropdown".
```

There are many tips in our Performance Tuning guidelines documentation for increasing performance when using the DatabaseBridge, such as having your application classes and database on the same machine to remove network latency and converting some COBOL I/O to ESQL in a few key places. You can find them all here.

DOCUMENTATION HIGHLIGHT

Which Version Is This Message Referring To?

Many informational and error messages displayed by isCOBOL include build and version numbers, but not the corresponding release version. For example, consider the output of the iscrun -info command, which might show something like “compiled with isCOBOL build #1145, minimum required #1142.” Or take an error such as:

“PROGRAM has been compiled by a more recent version of the Java Runtime (class file version 55.0); this version of the Java Runtime only recognizes class file versions up to 52.0.”

Another example is a client/server mismatch:

“Client release 74 is incompatible with Application Server 99.”

What Java or isCOBOL release do these numbers refer to?

To resolve this, the documentation provides a mapping of version and build numbers to their corresponding isCOBOL releases. You can find this list in isCOBOL Evolve : Appendices : Version numbers.

isCOBOL Debugging Techniques

Each COBOL developer has a routine for that most-common of tasks, debugging a program. Here are some common techniques that we suggest here at Veryant. Let us know if you have any other methods that you use.

Fully understand the problem.

We often make assumptions about where or what the error is based on a quick description of the outcome. Keep an open mind and make sure you have a good grasp of the problem before trying to fix it.

Backtracing

Debug backwards, starting from the point where the problem first began. You will probably set and unset many breakpoints and run your application multiple times as you do this.

Use the debugger

isCOBOL comes with two debuggers, one integrated in the IDE and one you can use from the IDE or the command line. isCOBOL’s debugger is a graphical tool, but if you’re debugging a process running in a non-graphical environment, you can easily run the isCOBOL debugger remotely.

Breakpoints and stepping

Set a breakpoint in the debugger, or create a monitor of a variable so that the debugger stops when the monitor changes or reaches a specified value. Step through the code line by line, skipping paragraphs and programs when they aren’t part of the problem.

Isolating the code

Once you find the problem, or at least the problem code, it’s almost always better to isolate this code in a separate program for testing. Cut and paste code to a new program, or comment out blocks of code at a time to see which block causes the problem.

Rubber ducking (sometimes called plastic programming).

Explain the problem to someone else, even if the someone else is a fictional character. Talking the problem through or writing it down forces you to think critically about it, giving you new insights and solutions.

Log analysis

Create log files and read through them. There are different logging levels in isCOBOL, but a good one to start with is 15. Try adding this to your command line:

```
iscrun <other command line switches>
-J-Discobol.logfile=c:\temp\mylog.txt
-J-Discobol.tracelevel=15 <program name>.
```

Take breaks

Sometimes you end up going “down the rabbit hole” when debugging a problem, and it helps to take a step back, clear your head, and start over again.

Contact Veryant technical support

(support@veryant.com) BEFORE you get too frustrated. We’re here to help.

Evolution, without revolution

Contact Us

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

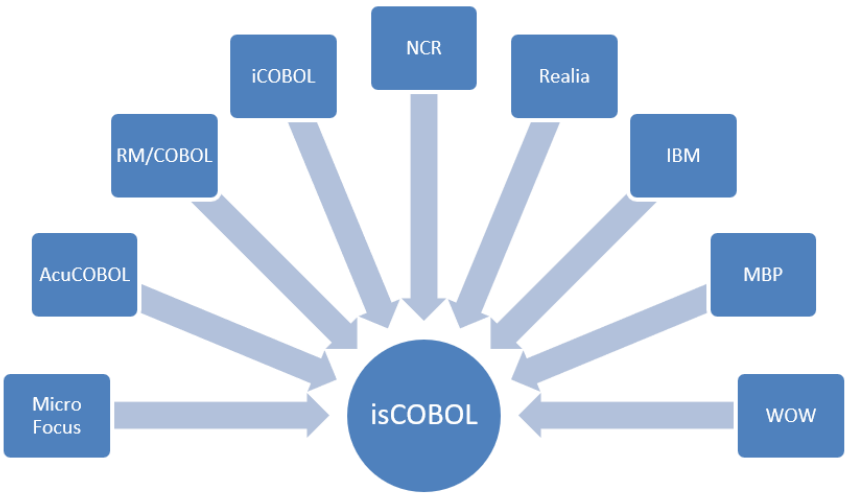
If you would like Veryant to contact you to schedule a technical product briefing, email us at **info@veryant.com**

If you would like Veryant to contact you for special quote or sales assistance 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, the newest isCOBOL Evolve release 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