



isCOBOL™ Evolve

isCOBOL Evolve 2024 Release 1 Overview

Copyright © 2024 Veryant LLC.

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and recompilation. No part of this product or document may be reproduced in any form by any means without the prior written authorization of Veryant and its licensors if any.

Veryant and isCOBOL are trademarks or registered trademarks of Veryant LLC in the U.S. and other countries. All other marks are the property of their respective owners.

isCOBOL Evolve 2024 Release 1 Overview

Introduction

Veryant is pleased to announce the latest release of isCOBOL Evolve, isCOBOL Evolve 2024 R1.

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

The new version supports the latest Java LTS release 21.

isCOBOL IDE is now based on a newer Eclipse version, 2023-09 (4.29).

isCOBOL Evolve 2024 R1 supports Jakarta in all applications running in web environments.

Starting from this release, it's easier to port an application written for a specific RDBMS to another without changing source code.

Details on these enhancements and updates are included below.

Java 21 support

isCOBOL Evolve and isCOBOL SDK 2024R1 are certified for Java 21, the current Java LTS (Long Term Support) release. It is important to have isCOBOL Evolve 2024R1 qualified to be used in the latest LTS version of Java because LTS applies the tenets of reliability engineering to the software development process and software release life cycle. Long-term support extends the period of software maintenance; it also alters the type and frequency of software updates (patches) to reduce the risk, expense, and disruption of software deployment, while promoting the dependability of the software.

We at Veryant feel it's important to test and qualify our products for Java LTS releases, and our internal tests are now performed on all 4 Java LTS versions: 1.8, 11, 17 and 21 to ensure the correct execution on each. Our users can then decide if and when to upgrade to a specific Java version, depending on their needs. Typically, COBOL programs function seamlessly across various Java versions. However, if these programs require interaction with third-party Java classes or libraries, it may become essential to utilize a specific version of the Java runtime. Having our products certified for any Java LTS version ensures that our users can move freely to the best available option.

One of the changes related to supporting Java 21 is the default file encoding, which is now UTF-8 on Windows platforms. If a COBOL application needs to support a different file encoding, it can be forced using the flag `-Dfile.encoding` startup option.

isCOBOL IDE enhancements

The isCOBOL IDE 2024 R1 is now based on Eclipse 2023-09 (4.29), which includes full support for Java 21. The new version offers many improvements in performance, stability and usability on a day-by-day usage. The installers include an embedded JRE for the IDE, thus requiring no additional downloads from the developer.

Below is an extract of the new functionalities:

The Text Editors support multiple selections that allow most edit operations (text replacement or insertion, extend selection to next word or next line, copy / paste,...) to apply simultaneously on all ranges.

Multiple selection can be enabled by:

- Turning a block selection into multi-selection using the “To multi-selection” command
- Adding a caret with Alt-click.
- Using the new “Select All” button on the Find/Replace dialog.

The windows title bar in the dark theme on Microsoft Windows is now styled in the default dark theme.

As of Java 21 the IDE uses the UTF-8 default file encoding on Windows platforms as well as the previously supported platforms. If a COBOL project needs to support a different file encoding, it can be changed in the project properties.

Jakarta support

Jakarta EE, formerly Java Platform, Enterprise Edition (Java EE) and Java 2 Platform, Enterprise Edition (J2EE), is a set of specifications extending Java SE with specifications for enterprise features such as distributed computing and web services. Jakarta EE applications are run on reference runtimes, which can be microservices or application servers. These handle transactions, security, scalability, concurrency and management of the components they are deploying.

JEE refers to Java servlet version 4, while Jakarta refers to Java servlet version starting from 5, and the class package name changes from javax to jakarta.

To run the newer servlet version 5, you need a newer Java container. This means Tomcat up to version 9 supports servlet version 4 while Tomcat 10 supports servlet version 5. Also, GlassFish 5 supports up to servlet version 4 while GlassFish 6 extends servlet support to version 5.

Veryant products now support servlet version 5 to allow deploying applications to the latest versions of Java Containers such as Tomcat and GlassFish.

isCOBOL 2024 R1 supports Jakarta in all our products that run in a web environment, such as isCOBOL WebClient, isCOBOL EIS Servlets, WebServices and WebDirect.

IsCOBOL WebClient

WebClient supports the Jakarta specification when it is run as a web application inside a Java Container, and will automatically activate the needed components in the weclient-server.war and webclient-admin-server.war files accordingly.

When WebClient is executed with the provided wrappers, it will use the embedded Jetty server.

IsCOBOL EIS

The new Jakarta servlet specification is supported in isCOBOL EIS Servlets and WebServices when using the standard “web.xml” file available with the installation.

If you need to deploy on an older version of a Java Container, a file called “web.jee.servlet.xml” is provided that contains the entries needed, and is the same as in previous releases.

The main difference between the two files are the lines that refer the class name. The following lines are needed to enable JEE support:

```
...
<filter>
  <filter-name>isCOBOL filter</filter-name>
  <filter-class>com.iscobol.web.IscobolFilter</filter-class>
</filter>
...
<servlet>
  <servlet-name>isCobol</servlet-name>
  <servlet-class>com.iscobol.web.IscobolServletCall</servlet-class>
</servlet>
...
<listener>
  <listener-class>com.iscobol.web.IscobolSessionListener</listener-class>
</listener>
...
```

The following lines are needed to support the new Jakarta specification:

```
...
<filter>
  <filter-name>isCOBOL filter</filter-name>
  <filter-class>com.iscobol.webjakarta.IscobolFilter</filter-class>
</filter>
...
<servlet>
  <servlet-name>isCobol</servlet-name>
  <servlet-class>com.iscobol.webjakarta.IscobolServletCall</servlet-class>
</servlet>
...
<listener>
  <listener-class>com.iscobol.webjakarta.IscobolSessionListener</listener-class>
</listener>
...
```

No other changes are needed in other deployed libraries, since iscobol.jar contains all the necessary classes.

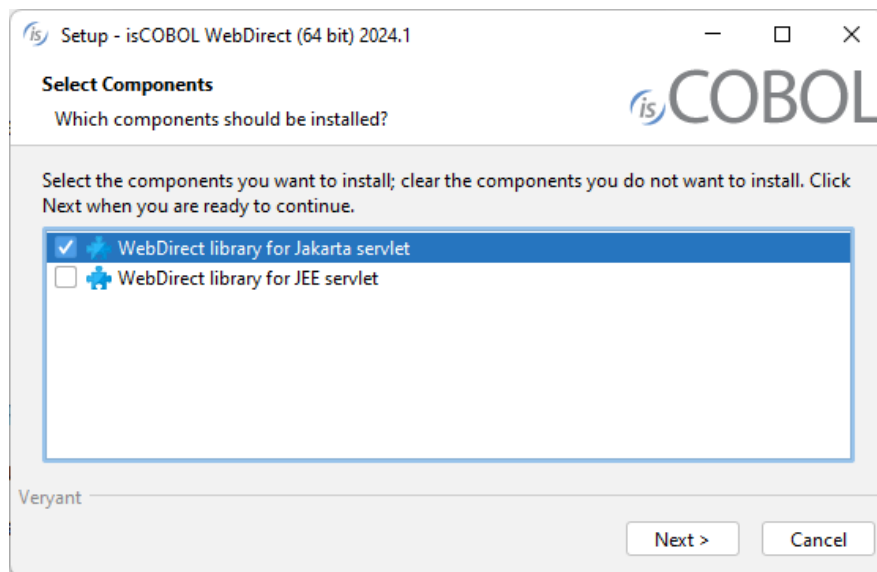
isCOBOL WebDirect

To support both JEE and Jakarta containers, two sets of libraries are provided for isCOBOL EIS WebDirect deployment that are needed by the underlying ZK libraries on which WebDirect is built. isCOBOL 2024 R1 provides a separate WebDirect installer that allows you to choose the set of libraries to install based on the container specification. The installer will deploy the selected libraries in the isCOBOL SDK main folder.

If both JEE and Jakarta libraries are chosen, the installer creates two additional subfolders in the isCOBOL SDK Webdirect folder. The “webdirect/lib” and “webdirect/xml” folders contain the necessary files for a Jakarta container, while the JEE related files will be stored in the “webdirect/lib/jee” and “webdirect/xml/jee”.

As shown in Figure 1, *isCOBOL WebDirect installer*, the developer can select the appropriate Jakarta or JEE servlet libraries depending on the container used for deploying.

Figure 1. isCOBOL WebDirect installer.

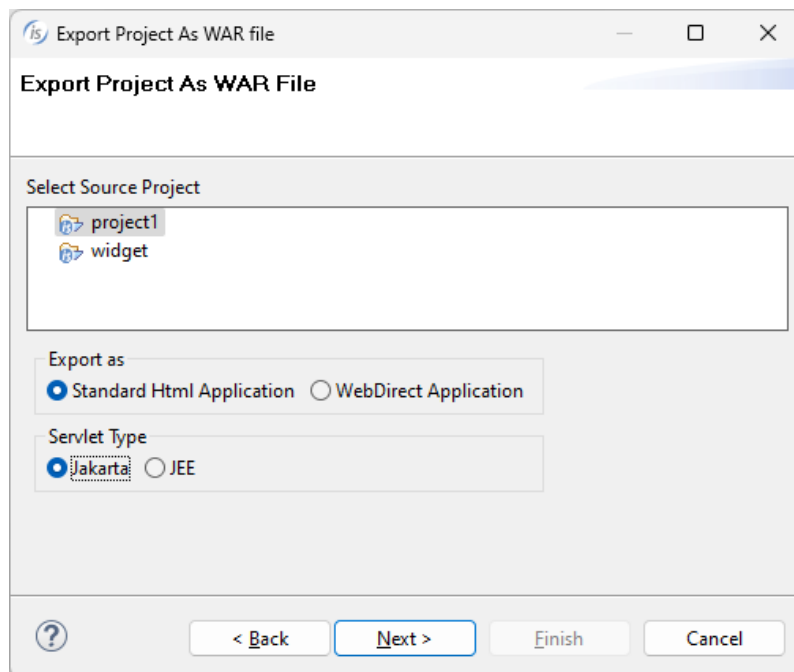


IsCOBOL IDE

isCOBOL 2024 R1 IDE supports Jakarta, and the feature “Export Project as WAR file” that creates a .war file to be deployed in a Java container now asks the user to select the Servlet Type and will generate the correct libraries and configuration for the selected scenario.

This feature is available for isCOBOL projects that use WebDirect and for projects that use the isCOBOL HTTPHandler class, for example applications that leverage the WebServices bridge generation. As shown in Figure 2, *Export to WAR file*, the developer can select Jakarta or JEE in the Servlet Type area depending on the container used when deploying the application.

Figure 2. Export to WAR file.



ESQL enhancements

IsCOBOL 2024 R1 contains many enhancements in the ESQL area. Use SQLJ code to improve performances in batch elaborations with a new compiler option. Simplify the work to move a COBOL application from one database to another with new configurations and interfaces. In addition, the compatibility with DB2prep has been improved.

SQLJ support

The isCOBOL compiler can now generate SQLJ code to be used instead of standard JDBC APIs on Oracle and DB2 databases. SQL is a non-procedural language for defining and manipulating data in relational databases. SQLJ is a language that embeds static SQL in Java in a way that is compatible with Java's design philosophy. SQLJ does syntax-checking of the embedded SQL, type-checking to assure that the data exchanged between Java and SQL have compatible types and proper type conversions, and schema-checking to assure congruence between SQL constructs and the database schema.

Using SQLJ instead of standard JDBC APIs on Oracle and DB2 databases provides better performance of the static SQL, including every DML query that is not prepared. To generate program classes that manage ESQL via SQLJ instead of standard JDBC APIs, add the `-sqlj` option to your compiler command line. For a correct result the "sqlj" command (the SQLJ translator) must be available in the Path environment. To see the difference between standard JDBC APIs and SQLJ, you can use `-jj -jc` in the compiler command line and look at the intermediate java source.

A query like this:

```
EXEC SQL
  INSERT INTO TBL (COD, NAME, ADDRESS)
    VALUES (:WK-COD, :WK-NAME, :WK-ADDR)
END-EXEC.
```

is translated to a call to the isCOBOL's `ESQLRuntime` based on JDBC APIs as:

```
ESQL_CURS_HNDL.set(Esql.DECLARE_STMT(new Object[] {
    SQLCA,null, null, "INSERT INTO TBL(COD, NAME, ADDRESS) VALUES ( ?, ?, ?)", null, "0", null, "0"}));
RETURN_CODE.set(Esql.SET_PARAM(new Object[] {
    ESQL_CURS_HNDL, WK_ADDR,ESQL_BIND_TYPE,Factory.getNumLiteral(3, 4, 0, false),null, null, $0$, "INOUT"}));
RETURN_CODE.set(Esql.SET_PARAM(new Object[] {
    ESQL_CURS_HNDL, WK_NAME,ESQL_BIND_TYPE,Factory.getNumLiteral(2, 4, 0, false),null, null, $0$, "INOUT"}));
RETURN_CODE.set(Esql.SET_PARAM(new Object[] {
    ESQL_CURS_HNDL, WK_COD,ESQL_BIND_TYPE,Factory.getNumLiteral(1, 4, 0, false),null, null, $0$, "INOUT"}));
RETURN_CODE.set(Esql.EXECUTE(new Object[] {
    SQLCA, ESQL_CURS_HNDL}));
```

When the `-sqlj` switch is used the query is translated to this code:

```
#sql [ctxINS] {INSERT INTO TBL(COD, NAME, ADDRESS)
  VALUES ( :sqlj$p$WK_COD, :sqlj$p$WK_NAME, :sqlj$p$WK_ADDR)};
```

To compile and run the isCOBOL's ESQL sample on DB2 with SQLJ, follow these steps:

1. Compile the program with the `-sqlj` option, e.g.:

```
iscc -sqlj ESQL-SAMPLE.cb1
```

2. Bind the generated profile file to the database, e.g.

```
db2profcb -url " jdbc:db2://localhost:50000/SAMPLE" -user db2inst1 -password
secret ESQL_SAMPLE_SJProfile0.ser
```

3. Run the program:

```
iscrun ESQL_SAMPLE
```

A table of performance gains is shown in Figure 3, *Comparing JDBC vs SQLJ*, where the same program is executed with both the standard JDBC and SQLJ. The test was run in 64-bit Ubuntu Linux on an Intel Core i5 Processor 4440+ clocked at 3.10 GHz with 16 GB of RAM, using Oracle JDK 1.8.0_381 and IBM DB2 11.5. Times are expressed in seconds. The COBOL program reads 96,000 rows from a table using single SELECT statements:

```
EXEC SQL
  SELECT field-1, field-2, ..., field-n
  INTO :hostvar-1, :hostvar-2, ..., :hostvar-n
  FROM table-name WHERE id = ?
END-EXEC
```

The test has been repeated 3 times to get an average time.

Figure 3. Comparing JDBC vs SQLJ.

Iteration	N. queries	Time JDBC	Time SQLJ
1	96,000	28.50	17.91
2	96,000	29.43	18.50
3	96,000	28.66	19.10
Average	96,000	28.86	18.50

Move to a different Database

When COBOL application written with ESQL code for a specific RDBMS need to be migrated to a different RDBMS, developers face the challenge of adjusting the SQL code that may have been written to support a specific RDBMS and is not compatible with another RDBMS. isCOBOL 2024 R1 can ease the migration with both the improved PreProcessor, which performs manipulation of Embedded SQL statements enclosed in the “EXEC-SQL” and “END-EXEC” statements, and the new configuration options and interfaces to customize handling of ESQL by the runtime framework.

The first step is to write a PreProcessor class using either Java or COBOL, and use it during the compilation to adjust the static ESQL statements used in the source. To use the PreProcessor it’s necessary to add the PreProcessor class name in the compiler configuration file.

For example, the query:

```
EXEC SQL
  SELECT CHAR(CURRENT DATE, ISO)
         INTO :wrk-date
         FROM SYSIBM.SYSDUMMY1
END-EXEC.
```

uses valid syntax in DB2 to retrieve the current date, but it is not supported by Oracle. The PreProcessor needs to change the code to:

```
EXEC SQL
  SELECT TO_CHAR(CURRENT_DATE, 'YYYY-MM-DD')
         INTO :wrk-date
         FROM DUAL
END-EXEC.
```

The new query can run on Oracle using its built-in TO_CHAR function instead of DB2’s CHAR, and the parameters passed to the function are different. Also the FROM clause needs to be changed to be supported by Oracle. The PreProcessor code needed to perform this change is provided as a sample in the isCOBOL installation.

The second step is to manage the same syntax replacement for dynamically generated queries, since those can’t be identified in the source or are not present in the source code at all. A new configuration named `iscobol.esql.prepare_handler` has been

implemented to allow developers to provide a class that customizes the PREPARED ESQL statements before they are executed. For example, the following code:

```
MOVE "SELECT CHAR(CURRENT DATE, ISO) FROM SYSIBM.SYSDUMMY1"  
    TO WRK-QUERY.  
EXEC SQL  
    PREPARE CMD FROM :wrk-query  
END-EXEC.
```

executes the same query shown before but with the PREPARE statement. It will be intercepted by the class that implements the new interface, which can then perform the needed changes to be compatible to the target RDBMS. The class must implement the interface [com.iscobol.rts.EsqlPrepareHandler](#), which requires the following method to be defined:

```
public void queryDecoder(CobolVar query)
```

The queryDecoder method allows you to alter the SQL query text of a prepared ESQL statement before it is executed and the same logic implemented in the previous PreProcessor class can be reused. To have your class be automatically called after each ESQL PREPARE, set `iscobol.esql.prepare_handler=<classname>` in the configuration. The installed sample also demonstrates how to use this configuration in conjunction with the PreProcessor code.

Different RDBMSs can return different status codes, so the last step involves the configuration of SQLCA fields to map the error codes to common codes. The new interface [com.iscobol.rts.EsqlSqlcaHandler](#) can ease this task by implementing the following method:

```
public void sqlcaDecoder(SQLException ex,  
                        CobolVar sqlcode,  
                        CobolVar sqlstate,  
                        CobolVar sqlerrmc)
```

You can use this method to set the SQLCODE, SQLSTATE and SQLERRMC fields before they're returned to the COBOL program. The method receives as input the instance of `java.sql.SQLException` that was raised then running the query, and you can inquire to get the error details and set SQLCA fields accordingly.

To have your class automatically called after each ESQL error, set `iscobol.esql.sqlca_handler=<classname>` in the configuration.

This feature integrates the existing `iscobol.esql.sqlcode.<value>=<new-value>` configuration setting.

With `iscobol.esql.sqlcode.<value>=<new-value>` you can map a SQLCODE value to another, creating a compatibility between different databases. But `iscobol.esql.sqlcode.<value>=<new-value>` cannot be used with databases like PostgreSQL, where the SQLCODE is 9999 for every error. In this case, you can set SQLCODE according to the SQLException with a custom class implementing the `com.iscobol.rts.EsqlSqlcaHandler` interface.

DB2 compatibility

COBOL programs can now expose ESQL cursors to their callers, who can now access the cursor data through an array of `java.sql.ResultSet` objects. This feature is particularly useful for sharing sets of data with Java programs that call COBOL programs using the `com.iscobol.java.IsCobol` class, with or without the EasyLinkage facility.

isCOBOL now supports the “WITH RETURN” clause in the DECLARE CURSOR statement, and you can define a cursor as follows:

```
EXEC SQL
  DECLARE sharedcur CURSOR FOR
    SELECT CLI_COD, CLI_NAME, CLI_ADDRESS
    FROM CLIENTS_TBL
    WITH RETURN
END-EXEC
```

The COBOL program should just open the cursor and leave it open, without doing any FETCH on it.

In the Java program, two new methods are available in the `com.iscobol.java.IsCobol` class:

public void registerResultSets() : this method must be called before calling the COBOL programs and instructs the isCOBOL Framework to collect all the cursors that were declared with the WITH RETURN clause that have been opened and have not been closed.

public ResultSet[] getResultSets() : this method can be called after calling the COBOL programs and returns the array of `ResultSet` objects that store cursors data. For every COBOL cursor collected by the Framework a `java.sql.ResultSet` is returned. The Java program can scan these `ResultSet` objects to get the data. When done, it can close these `ResultSet` to release allocated memory.

Compiler enhancements

The Compiler supports new syntax to allow inline variable declaration and improved the auto-boxing in Object Oriented Programming. A new compiler option has been implemented to create an external DataMap that declares all data items of DATA DIVISION.

Inline variable declaration

Many languages such, as C# and Java, support the inline variable declaration syntax that allows you to declare a local variable directly in a code block. This has different advantages, such as:

- no risk of creating conflicts by reusing the same variable in a different code block
- thread safe code related to the variable when running multi-threading programs
- fast code in loops since the inline variables use primitive types

The newest isCOBOL compiler implements primitive type inline variable declaration in the AS clause of PERFORM VARYING statements. For example, code like this:

```
perform varying in-idx as "int" from 1 by 1 until in-idx > 10000
    move my-var(in-idx) to back-var(in-idx)
    ...
end-perform
```

can be used to declare the in-idx variable, which is not declared in the DATA DIVISION but can be accessed in the code block between PERFORM VARYING and END-PERFORM.

It is basically the equivalent of, but performs better than, the following:

```
working-storage section.
01 w-idx      object reference "int".
...
    perform varying w-idx from 1 by 1 until w-idx > 10000
        move my-var(w-idx) to back-var(w-idx)
        ...
    end-perform
```

The inline variable type declaration follows the same syntax as the OBJECT REFERENCE clause and can reference a Java class name or a logical call name defined in the REPOSITORY.

Improved auto-boxing

Auto-boxing refers to the capability of mixing Java and primitive types with COBOL variables in the same statement, making source code more readable and flexible. Auto-boxing has been vastly improved in the 2024 R1 release by supporting:

- the comparisons between numeric Java data types and COBOL. For example, code such as:

```
working-storage section.  
77 obj-int      object reference "int".  
77 obj-long     object reference "long".  
77 obj-lang-int object reference "java.lang.Integer".  
77 obj-lang-long object reference "java.lang.Long".  
77 var-num      pic 9(5)v9(3).  
...  
    if obj-int > 10  
    if obj-int = var-num  
    if var-num < obj-lang-long  
    perform varying var-num from 1 by 1 until var-num > obj-long  
    perform parag2 until obj-lang-int = var-num  
    ...
```

can now be compiled and executed, making it simpler to compare a numeric COBOL data item with other Java data types and primitive types.

- COBOL arithmetic operations on primitive data types and objects. The COBOL statements ADD, SUBTRACT, MULTIPLY, DIVIDE and COMPUTE can now contain Java data types without limitations. For example, code such as:

```
add 1 to obj-int  
subtract var-num from obj-long  
multiply obj-lang-int by var-num giving obj-lang-long  
divide obj-lang-long by obj-lang-int giving var-num  
compute obj-lang-long = (var-num * 2) + 1
```

can be used to mix COBOL variables and Java data types in the same statement.

- arithmetic operations in CALL and INVOKE parameters. This is typically useful when passing parameters BY VALUE in CALL and when passing numeric parameters to Java methods. For example, the following code is now allowed:

```
call "PROG" using by value var-num - 1  
                by value var-num + 1  
if obj-string:>substring(var-num + 1, var-num + 2) = "A"
```

External DataMap

The new compiler option `-edm` can create an xml file with the application's DataMap. This file contains all the fields declared in DATA DIVISION; describing the details, declaration location and usage information. This xml file can be used to perform analysis with external tools or can be imported in a RDBMS system and queried. It can also be used to generate documentation by developers, for example.

Using the compiler option `-edo=<path>` you can specify the location for the DataMap output file.

For example, compiling with this command:

```
iscc -edm -edo=datamap progcust.cbl
```

a source that contains:

```
working-storage section.  
77 V1          pic 99.  
01 G1-VARS.  
   03 G1-V1    pic 99.  
   03 G1-V2    pic x(10).  
linkage section.  
copy "def-params.def".
```

with the copyfile `def-params.def` containing:

```
01 DEF-PAR1    pic x(10).  
01 DEF-PAR2.  
   03 DEF-PAR-V3 pic 9(5).
```

This xml file can be used by an external tool to inquire all the information relative to every data-item; such as name, offset, length, type and declaration location (FD, Working, Linkage, ...). Additional details include whether the field is elementary, boolean, constant, occurs, redefines, external, used by the program, or used as parameter in the USING clause of CALL and INVOKE statements. You can also determine if the data-item is declared in the main source or in a COPY file and also the replacing clause in case of a COPY REPLACING.

The file progcust.xml is created in the datamap folder with the following content:

```
<program name="PROGCUST">
  <field>
    <name>V1</name>
    <location>WS</location>
    <offset>0</offset>
    <physicalLength>2</physicalLength>
    <dataType>NumUnsigned</dataType>
    <elementary>yes</elementary>
    <usedByProgram>yes</usedByProgram>
    <usedAsParameter>yes</usedAsParameter>
  </field>
  <field>
    <name>G1-VARS</name>
    <location>WS</location>
    <offset>0</offset>
    <physicalLength>14</physicalLength>
    <dataType>Alphanum</dataType>
    <group>yes</group>
    <usedByProgram>no</usedByProgram>
    <usedAsParameter>no</usedAsParameter>
  </field>
  <field>
    <name>G1-V1</name>
    <location>WS</location>
    <offset>0</offset>
    <physicalLength>2</physicalLength>
    <dataType>NumUnsigned</dataType>
    <elementary>yes</elementary>
    <usedByProgram>no</usedByProgram>
    <usedAsParameter>no</usedAsParameter>
  </field>
  <field>
    <name>G1-V2</name>
    <location>WS</location>
    <offset>2</offset>
    <physicalLength>10</physicalLength>
    <dataType>Alphanum</dataType>
    <elementary>yes</elementary>
    <usedByProgram>no</usedByProgram>
    <usedAsParameter>no</usedAsParameter>
  </field>
  <field>
    <name>DEF-PAR1</name>
    <location>LS</location>
    <offset>0</offset>
    <physicalLength>10</physicalLength>
    <dataType>Alphanum</dataType>
    <elementary>yes</elementary>
    <usedByProgram>yes</usedByProgram>
    <usedAsParameter>no</usedAsParameter>
    <copyFile>p2-params.def</copyFile>
  </field>
  ...
</program>
```

Compatibility enhancements

IsCOBOL 2024 R1 improves compatibility with IBM COBOL by supporting LINE LIMIT in the REPORT SECTION and by implementing additional functions.

Report Section

The Report Section is a COBOL section supported by IBM that allows definition of a report. It produces a text file with the statements “initiate”, “generate” and “terminate”.

isCOBOL 2024R1 adds support for the LINE LIMIT clause to specify the maximum number of characters that can be written to a line. Characters exceeding the line limit are truncated from the output.

For example, this code:

```
report section.  
rd my-report  
  control are final  
  page limit 22 lines  
  line limit 50  
  heading 1.  
01 detail-line type de line plus 2.  
  02 line plus 1.  
    03 column 4 pic x(10) source field1.  
    03 column 20 pic x(20) source field2.  
  ...  
open output F1.  
initiate my-report.  
...  
generate detail-line.  
terminate my-report.  
close F1.
```

sets the LINE LIMIT to 50. The runtime will truncate the output generated by the running program to the specified size.

New functions

NUMVAL and NUMVAL-C are existing functions meant to convert an alphanumeric variable to a numeric variable, and NUMVAL-C can be used on values containing currency symbols and/or commas.

To improve compatibility with IBM COBOL, the new NUMVAL-F function has been implemented in isCOBOL 2024 R1. This function extends conversion support to number strings that contain an exponent value.

Every function can be tested with the specific function named TEST-NUMVAL* that returns 0 if the argument passed conforms to the argument rules for the corresponding NUMVAL* function, or the position of the first character encountered that invalidates the string.

A code snippet like this:

```
if function test-numval(varx) = 0
  move function numval(varx) to varn
end-if
...
if function test-numval-c(varx) = 0
  move function numval-c(varx) to varn
end-if
...
if function test-numval-f(varx) = 0
  move function numval-f(varx) to varn
end-if
```

is now valid and can be compiled and executed successfully.

Runtime enhancements

The runtime 2024 R1 has been enhanced with new configuration options and a new library routine.

New configurations

When a COBOL application employs thread programming, leveraging statements like CALL THREAD or PERFORM THREAD, it becomes challenging to track the program's execution flow due to multiple threads running concurrently. While it is feasible to navigate through threads step by step in a debugger, there are instances when it is crucial to observe the simultaneous execution of all threads. This reflects the actual scenario when the program is running in a live production environment.

Analyzing runtime execution has been simplified with the introduction of a new configuration, namely `iscobol.logfile.thread=true`. This configuration enables the tracking of thread numbers at the end of each log line.

For example, a code such as:

```
perform thread DO-CHECK
perform thread DO-CHECK
...
DO-CHECK.
  call "$SLEEP" using 1
  set environment "myevn" to varx
  ...
```

when running using the following configuration settings:

```
iscobol.logfile=isc.log
iscobol.tracelevel=1039
```

produces the following content in the log file:

```
...
12... INFO: ENTER PARAGRAPH 'DO-CHECK' [PROGA]
12... INFO: ENTER PARAGRAPH 'DO-CHECK' [PROGA]
12... INFO: ENTER isCOBOL LIB '$SLEEP' {
12... INFO: ENTER isCOBOL LIB '$SLEEP' {
12... INFO: SET ENVIRONMENT [users] 'iscobol.myevn=cust2'
12... INFO: SET ENVIRONMENT [users] 'iscobol.myevn=cust1'
...
```

It's possible to see that the paragraph DO-CHECK has been executed 2 times, but it's unclear if it has been executed by the same thread or two different threads.

Adding the new configuration:

```
iscobol.logfile.thread=true
```

the output becomes:

```
...  
12... INFO: ENTER PARAGRAPH 'DO-CHECK' [PROGA] [PROGA Thread=4]  
12... INFO: ENTER PARAGRAPH 'DO-CHECK' [PROGA] [PROGA Thread=5]  
12... INFO: ENTER isCOBOL LIB 'C$SLEEP' { [com.iscobol.lib.C$SLEEP Thread=4]  
12... INFO: ENTER isCOBOL LIB 'C$SLEEP' { [com.iscobol.lib.C$SLEEP Thread=5]  
12... INFO: SET ENVIRONMENT [users] 'iscobol.myevn=cust2' [PROGA Thread=5]  
12... INFO: SET ENVIRONMENT [users] 'iscobol.myevn=cust1' [PROGA Thread=4]  
...
```

The log file now contains the thread ID that executed a statement, making it easier to follow the program flow.

A new configuration, `iscobol.terminal.paste_key`, allows users to customize the key used for pasting in character-based accept statements. By default, pasting is accomplished through a mouse right-click. With the new configuration, it is now possible to modify the key combination or add additional keys for the same action. For instance, by setting:

```
iscobol.terminal.paste_key=mrdrw,*p
```

Ctrl+P can be used for pasting in addition to the mouse right click.

New routine

A new routine named C\$PARAMNAME has been implemented to retrieve the name of the parameter passed from the caller program. This information can be useful if specific code needs to be executed based on the parameter.

The called program, if needed, could use the output generated by the new external DataMap using the -edm compiler option and apply custom logic based on the name, level, or structure of the parameter received by the calling program.

For example, if a caller program executes this code

```
77 MY-VAR          PIC X(20).
01 MY-GROUP.
   03 MY-VAR-1     PIC X(10).
   03 MY-VAR-2     PIC 9(6)V99.
   03 MY-VAR-3     PIC X(5).
...
CALL "PRCUST" USING MY-VAR, MY-GROUP
```

In the called program, the following code:

```
77 paramName pic x(30).
...
move 1 to paramNum
CALL "C$PARAMNAME" USING paramNum, paramName
                        GIVING ret-code
...
add 1 to paramNum
CALL "C$PARAMNAME" USING paramNum, paramName
                        GIVING ret-code
```

can be used to retrieve the string "MY-VAR" and "MY-GROUP", the parameter names specified in the caller program, in the paramName variable. The ret-code variable specified in the giving clause will contain 1 if the paramName can be set correctly, and will contain 0 if the specified parameter is a constant or was omitted.

GUI enhancements

IsCOBOL Evolve 2024 R1 GUI grid capabilities have been enhanced by implementing new properties to customize colors. A new property input-filter is now supported to better filter the input.

Grid colors

The grid control has been improved and now you can set the rollover row color. Setting the row-rollover-color or alternatively row-rollover-foreground-color and row-rollover-background-color properties in the grid control will make the entire row be painted with the specified colors when the mouse is over a cell.

When the mouse hovers in a header cell, the single cell is painted with the color properties specified in the new heading-rollover-color or heading-rollover-background-color and heading-rollover-foreground-color.

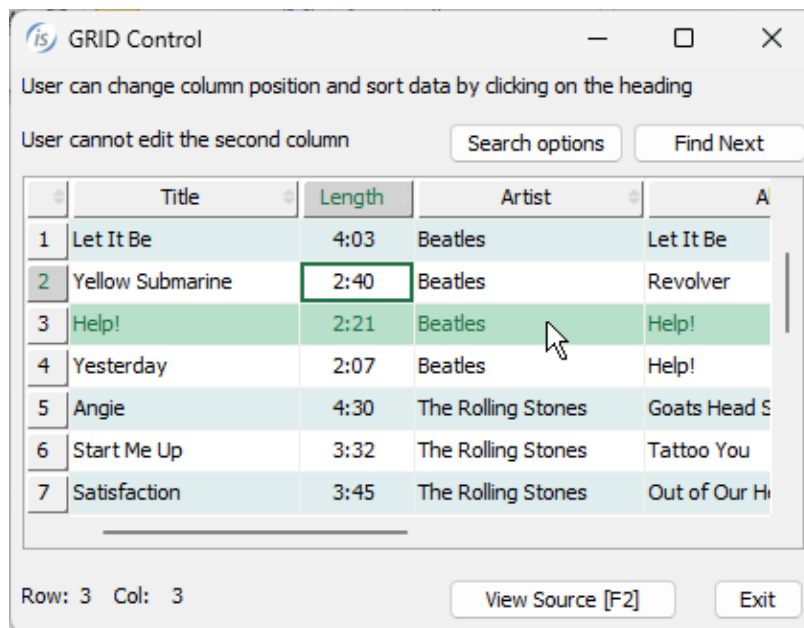
The new cursor-frame-color can be used to apply a specific color to the borders of a cell that has focus.

For example, this is the code where the colors of the grid are set:

```
03 Gd grid boxed vscroll
   column-headings row-headings centered-headings tiled-headings
   adjustable-columns reordering-columns sortable-columns
   row-background-color-pattern      (-16777215, -14675438)
   end-color                          -16774581
   heading-color                       257
   border-color                        rgb x#ACACAC
   heading-cursor-background-color     rgb x#D2D2D2
   heading-cursor-foreground-color     rgb x#217346
   cursor-frame-color                  rgb x#217346
   row-rollover-background-color      rgb x#B7DFC9
   row-rollover-foreground-color     rgb x#217346
   heading-rollover-background-color  rgb x#9FD5B7
   heading-rollover-foreground-color  rgb x#000000
   ...
```

The result of the program running is shown in Figure 4, *Grid colors*. The row covered by the mouse pointer is now more noticeable, and the cursor frame color has been harmonized to better blend with the colors used in the headings, providing clearer indications of the cursor's position.

Figure 4. Grid colors.



The screenshot shows a window titled "GRID Control" with a subtitle "User can change column position and sort data by clicking on the heading". Below the subtitle, it says "User cannot edit the second column" and there are two buttons: "Search options" and "Find Next". The main content is a table with the following data:

	Title	Length	Artist	A
1	Let It Be	4:03	Beatles	Let It Be
2	Yellow Submarine	2:40	Beatles	Revolver
3	Help!	2:21	Beatles	Help!
4	Yesterday	2:07	Beatles	Help!
5	Angie	4:30	The Rolling Stones	Goats Head S
6	Start Me Up	3:32	The Rolling Stones	Tattoo You
7	Satisfaction	3:45	The Rolling Stones	Out of Our Hi

At the bottom of the window, it displays "Row: 3 Col: 3" and two buttons: "View Source [F2]" and "Exit".

Input-Filter property

A new property named input-filter is now supported in controls that accept data input, such as the Entry-Field, Combo-Box and Grid, giving the developer the ability to better filter the text users can type. A regular expression is used in this property, making it extremely flexible.

A code snippet like this:

```
03 ef1 entry-field
   line 3 col 2 size 67 cells
   input-filter "[A-Za-z]+"
   ...
03 cb1 combo-box drop-down unsorted
   line 6 col 2 size 67 cells
   input-filter "[A-Z\sa-z]+"
   ...
03 Gd grid
   line 10 col 2 lines 9 size 67 cells
   display-columns (1, 5, 25, 35, 55, 80, 100, 120)
   data-columns    (1, 4, 34, 39, 59, 89, 104, 134)
   input-filter    ("*", "*", "[0-9:]+", "*",
                   "*", "*", "*", "[0-9]+")
   ...
```

defines different rules of acceptable input data in the controls:

- the entry-field accepts letters without spaces
- the combo-box accepts letters and spaces
- the grid accepts numbers and ":" in the third column and only numbers in the last column. The rest of the columns have no restrictions.

GIFE enhancements

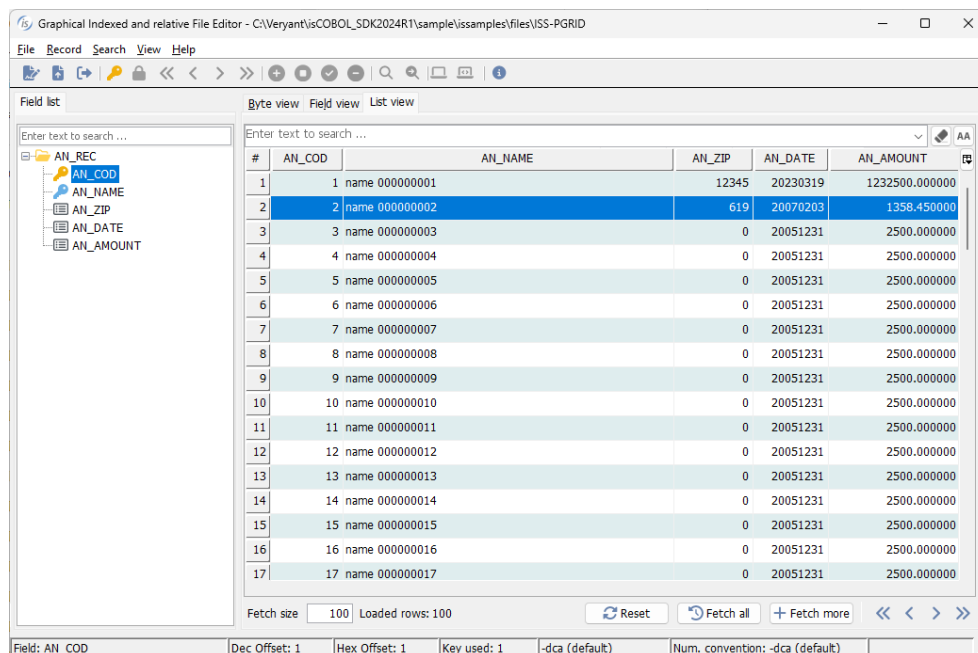
The GIFE (Graphical Indexed and relative File Editor) utility, has been improved in the 2024R1 release to add a new view that lets you see multiple records in a grid where every line is a record and every column is a field of the file. It also now supports files with 01 level redefines or conditional 01 levels in the FD declarations.

New List view

When an indexed or relative file along with the EFD file created by the compiler option – efd is opened by GIFE, a new view named List view is available to see multiple records in a grid. This view is read-only, so to change the content or insert a new record the previous Field view should be used.

As shown in Figure 5, *New List view*, the records are loaded in grid rows. Using the buttons on the bottom pane, additional records can be loaded and viewed.

Figure 5. New List view.



The search box can be used to filter the data and load only the records that meet the filter criteria.

Condition support

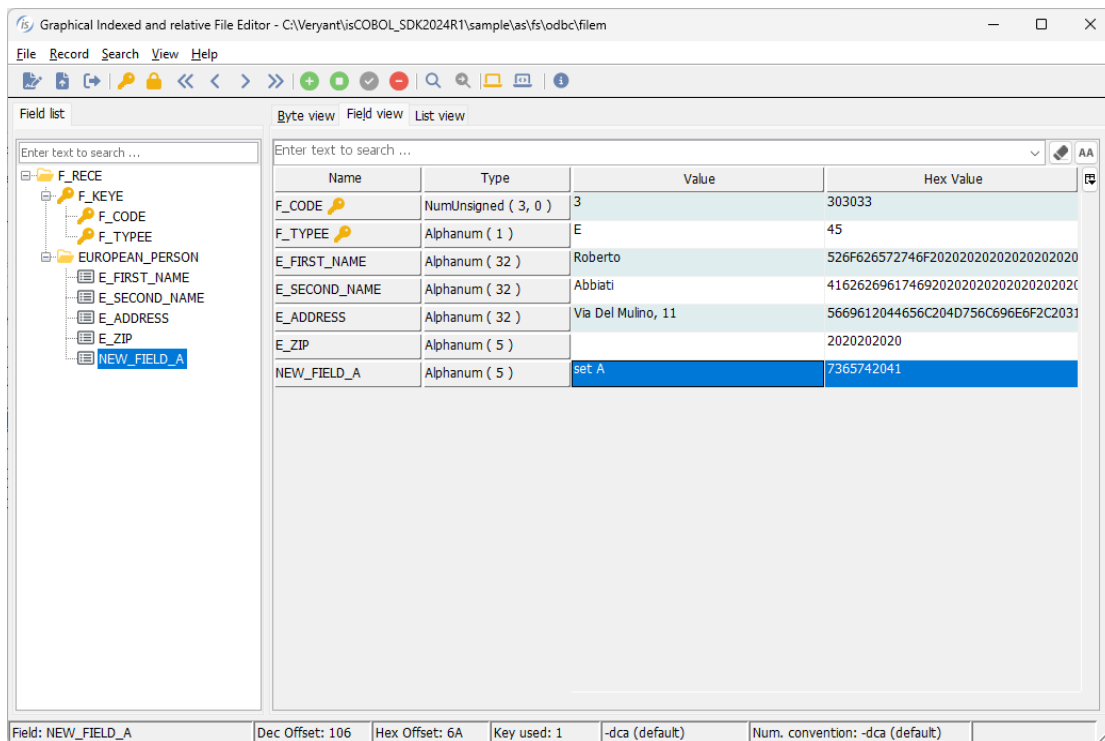
Veryant products such as Database Bridge and UDBC driver fully support files with multiple or conditional 01 levels in the field declaration. This information is stored in an .xml file generated using the -efd compiler option, and now GIFE can use this information to better display the records of such files in the Field view.

The following FD definition contains multiple 01 level definition, based on conditions:

```
fd filem.
$EFD WHEN F_TYPE = "M" TABLENAME = AMERICAN_PEOPLE
01 f-recM.
03 f-key.
05 f-cod          pic 9(3).
05 f-type        pic x.
03 american-person.
05 a-first-name  pic x(32).
05 a-second-name pic x(32).
05 a-address     pic x(32).
05 a-zip         pic x(5).
05 add-field-1  pic x(10).
05 add-field-2  pic x(10).
$EFD WHEN F_TYPE = "E" TABLENAME = EUROPEAN_PEOPLE
01 f-recE.
03 f-keyE.
05 f-codE        pic 9(3).
05 f-typeE       pic x.
03 european-person.
05 E-first-name  pic x(32).
05 E-second-name pic x(32).
05 E-address     pic x(32).
05 E-zip         pic x(5).
05 new-field-A   pic x(5).
05 filler        pic x(15).
...
```

and is now fully supported by GIFE.

Figure 8. Condition for European record.



It’s also possible to list all the conditions present in the file by opening the File Info and then pressing the button named “Conditions”. This opens an additional window as shown in Figure 9, *File Info*, and Figure 10, *Conditions*.

The same window with the list of conditions is also opened when inserting a new record that allows you to pick which 01 level to use when setting the fields contents.

Figure 9. File Info.

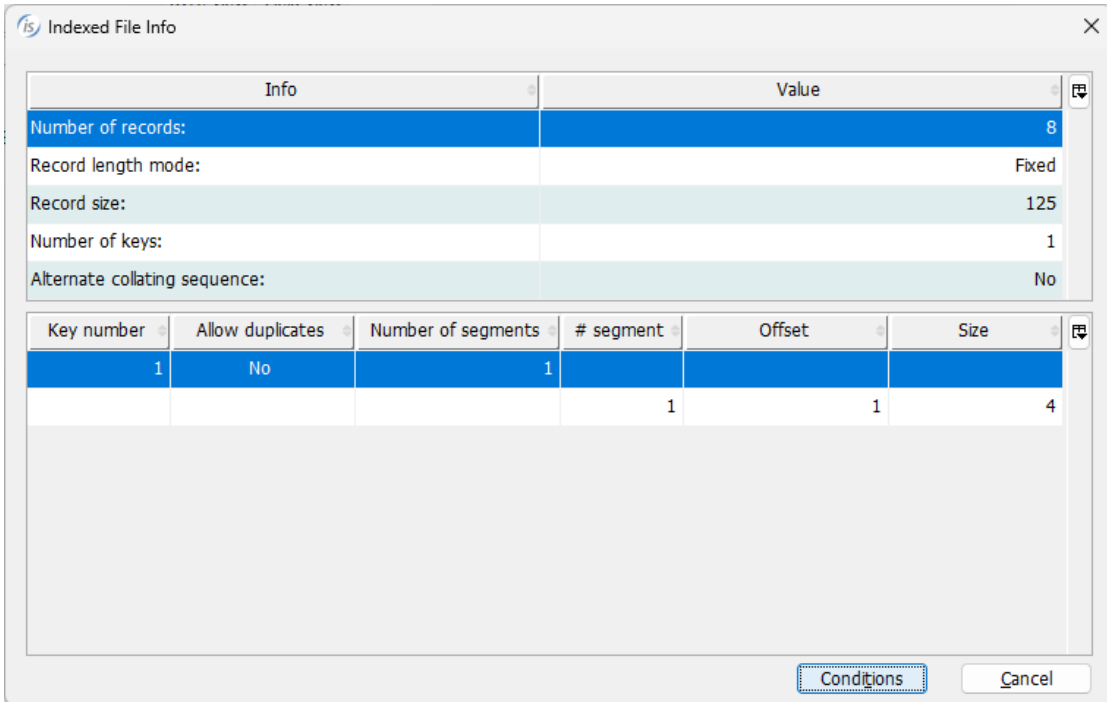


Figure 10. Conditions.

