isCOBOL Evolve: Database Bridge

Key Topics:

Table of Contents

Overview

This manual is intended for software developers who want to combine the reliability of COBOL programs with the flexibility and efficiency of a relational database management system (RDBMS). This manual gives systematic instructions on how to use DatabaseBridge, a program designed to allow for efficient management and integration of data with COBOL using the supported database engine.

DatabaseBridge (also known as EasyDB) generates EDBI COBOL program interfaces that provide a communication channel between COBOL programs and supported RDBMS.

The EDBI routine allows COBOL programs to efficiently access information stored in the RDBMS.

In order to store data, COBOL programs usually use standard indexed files. Information stored in indexed files is traditionally accessed through standard COBOL I/O statements like READ, WRITE and REWRITE.

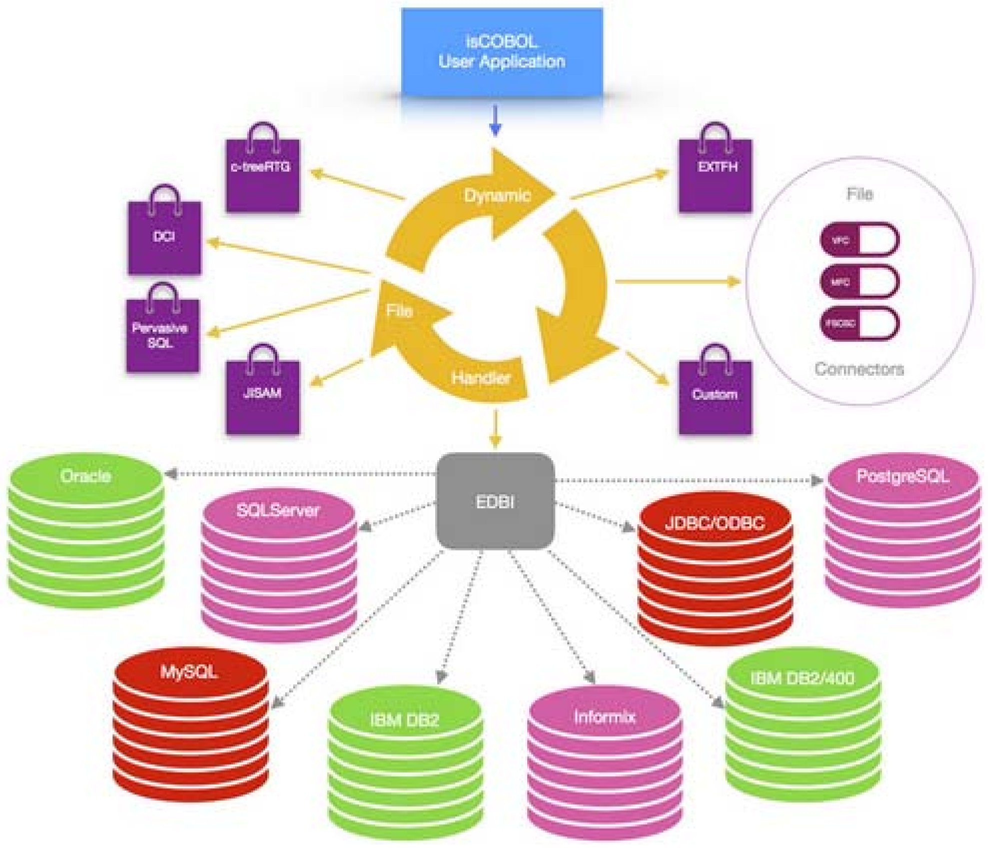
Traditionally, COBOL programmers use a technique called embedded SQL to embed SQL statements into the COBOL source code.

Though this technique is a good solution for storing information on a database using COBOL programs, it has some drawbacks. First, it implies COBOL programmers have a good knowledge of the SQL language.

Second, a program written in this way is not portable. In other words, it cannot work both with indexed files and the RDBMS. Furthermore, SQL syntax often varies from database to database. This means that a COBOL program embedding SQL statements tailored for a specific RDBMS might not work with another database. Finally embedded SQL is difficult to implement with existing programs. In fact, embedded SQL requires significant application re-engineering, including substantial additions to the working storage, data storage, and reworking the logic of each I/O statement.

DatabaseBridge, through the EDBI COBOL routine, allows the user to maintain existing COBOL code while accessing to the power of RDBMS. In this way, COBOL programmers need not be familiar with SQL and COBOL programs can stay portable with its indexed file system.

The isCOBOL dynamic file handler is used as a plug-in to the isCOBOL runtime, permitting management of data files from different file systems. With this feature, you can dynamically use different supported indexed file systems like c-tree and jisam or you can decide to use database management systems such as Oracle, DB2, DB2/400, Microsoft SQL Server, PostgreSQL, Informix and MySQL. Any ODBC/JDBC compliant RDBMS could be used with few limitations.



EDBI routines take care of different SQL syntax of supported RDBMS. EDBI routines are standard COBOL programs created at compile time from DatabaseBridge.

EDBI routines map COBOL fields into database fields adapting different COBOL data types into RDBMS data types and vice versa.

Getting Started

The setup of a DatabaseBridge environment requires the following steps:

1. Download and install the Java Development Kit (JDK)
2. Download and install isCOBOL Evolve SDK
3. Activate the License

In order to activate your isCOBOL Evolve products, you will need the e-mail you received from Veryant containing your license key. Contact your Veryant representative for details.

Download and install the Java Development Kit (JDK)

A JDK must be installed on your machine in order to use isCOBOL DatabaseBridge. For best results and performance, install the latest JDK version available for your platform. isCOBOL DatabaseBridge is certified to work correctly with both Oracle JDK and OpenJDK from version 8 to version 11.

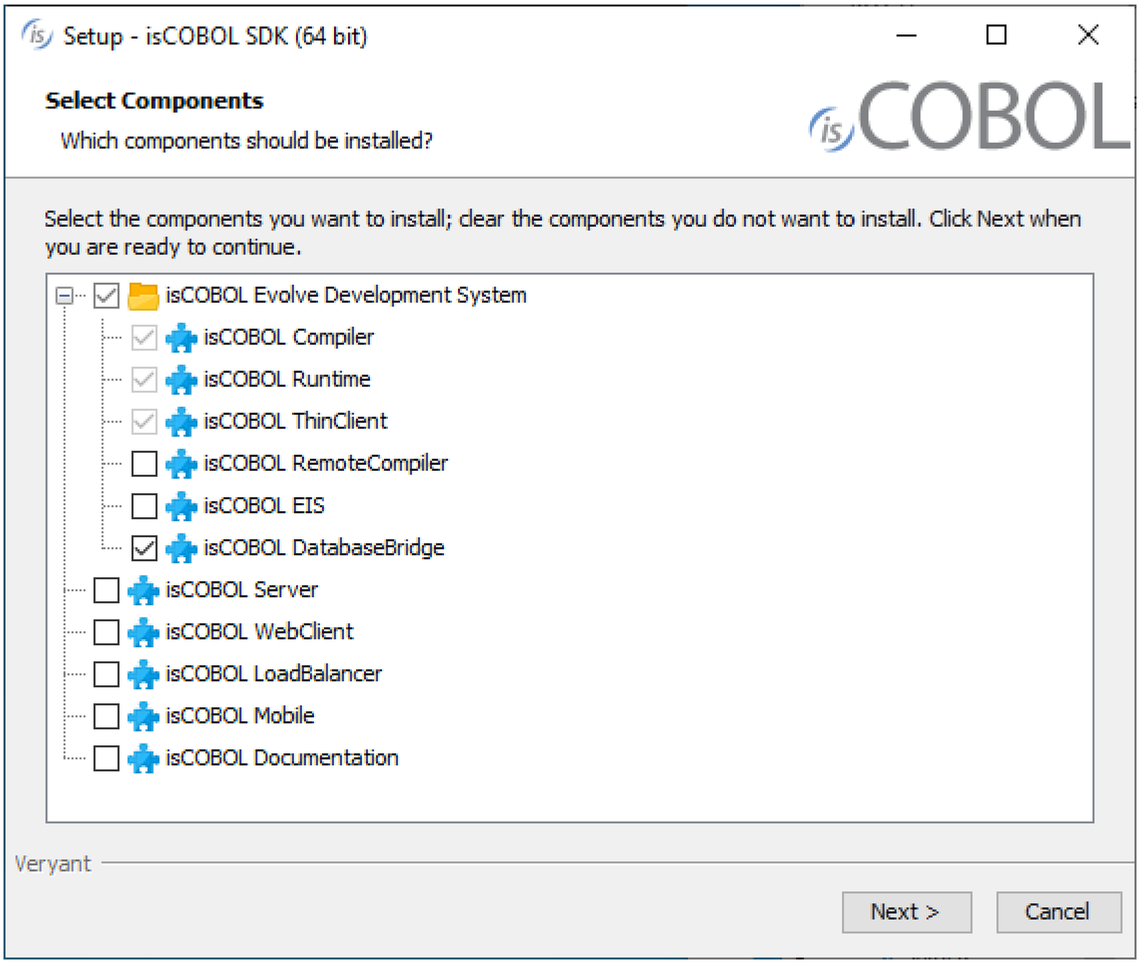
Self-extracting setups are provided for the Windows platform.

On Unix/Linux platforms Java may be already installed. If it’s not the case, you can install it using the appropriate system commands (e.g. yum, or apt-get).

Download and install isCOBOL Evolve SDK

Windows

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .
2. Go to "https://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the list of files for Windows x64 64-bit or Windows x86 32-bit. Select isCOBOL_2021_R1_*n*_Windows.*arc*.msi, where *n* is the build number and *arc* is the system architecture.
6. Run the downloaded installer to install the files.
7. Select "isCOBOL Compiler and Runtime Environment" and "isCOBOL DatabaseBridge" from the list of products when prompted.


Setup - isCOBOL SDK (64 bit)

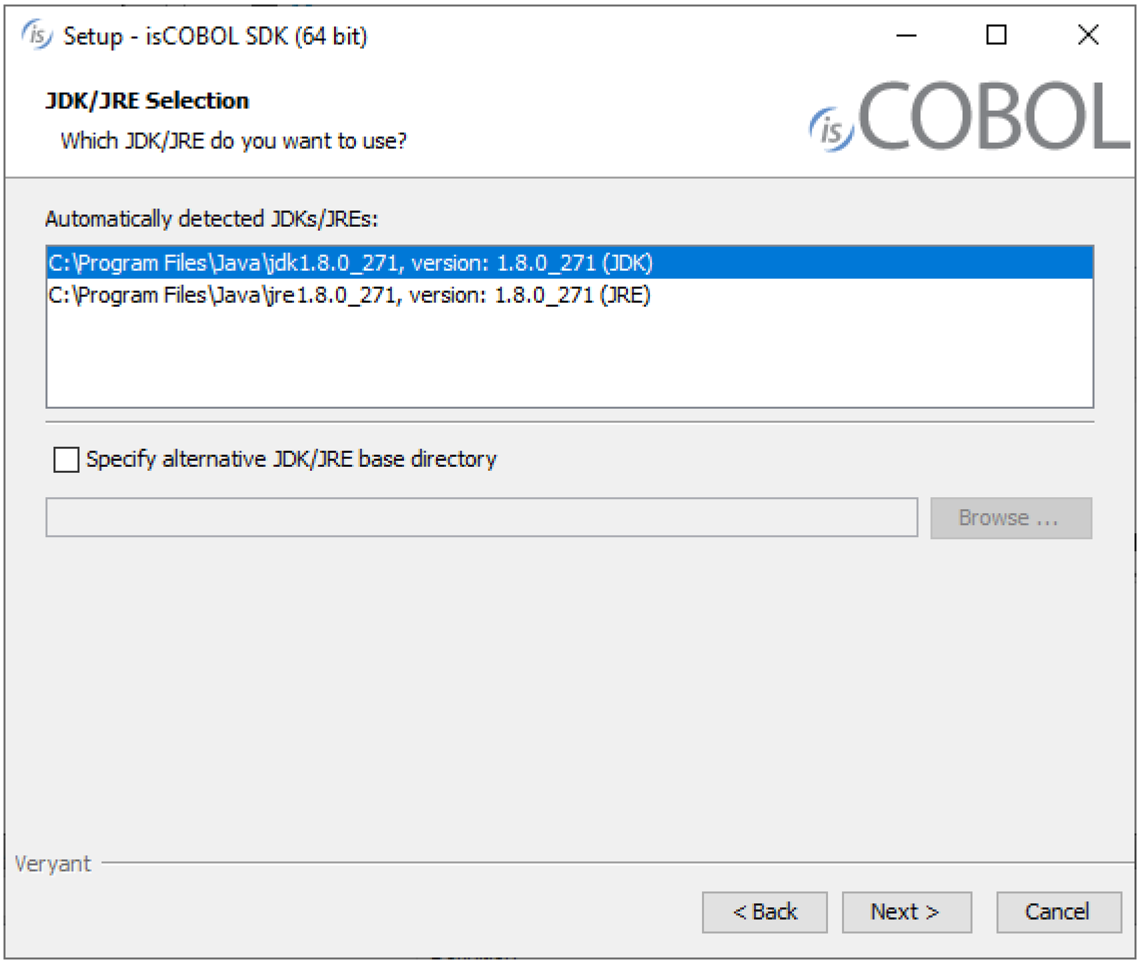
Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ✓ isCOBOL Evolve Development System
  - ✓ isCOBOL Compiler
  - ✓ isCOBOL Runtime
  - ✓ isCOBOL ThinClient
  - ☐ isCOBOL RemoteCompiler
  - ☐ isCOBOL EIS
  - ✓ isCOBOL DatabaseBridge
- ☐ isCOBOL Server
- ☐ isCOBOL WebClient
- ☐ isCOBOL LoadBalancer
- ☐ isCOBOL Mobile
- ☐ isCOBOL Documentation

8. Select your JDK when prompted


Setup - isCOBOL SDK (64 bit)

JDK/JRE Selection

Which JDK/JRE do you want to use?

Automatically detected JDKs/JREs:

C:\Program Files\Java\jdk1.8.0_271, version: 1.8.0_271 (JDK)
C:\Program Files\Java\jre1.8.0_271, version: 1.8.0_271 (JRE)

☐ Specify alternative JDK/JRE base directory

9. Follow the wizard procedure to the end. In the process you will be asked to provide the installation path ("C:\Veryant" by default) and license keys. You can skip license activation and perform it later, as explained in Activate the License.

Linux, FreeBSD, Mac OSX and SunOS

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .
2. Go to "https://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down, and select the appropriate .tar.gz file for the product and platform you require.
6. Extract all contents of the archive. For example,
   on Linux 32 bit:

```
gunzip isCOBOL_2021_R1_*_Linux.32.i586.tar.gz
tar -xvf isCOBOL_2021_R1_*_Linux.32.i586.tar
```

on Linux 64 bit:

```
gunzip isCOBOL_2021_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2021_R1_*_Linux.64.x86_64.tar
```

on FreeBSD:

```
gunzip isCOBOL_2021_R1_*_FreeBSD.64.tar.gz
tar -xvf isCOBOL_2021_R1_*_FreeBSD.64.tar
```

on Mac OSX:

```
gunzip isCOBOL_2021_R1_*_MacOSX.64.x86_64.tar.gz
tar -xvf isCOBOL_2021_R1_*_MacOSX.64.x86_64.tar
```

on SunOS:

```
gunzip isCOBOL_2021_R1_*_SunOS.64.tar.gz
tar -xvf isCOBOL_2021_R1_*_SunOS.64.tar
```

7. Change to the "isCOBOL2021R1" folder and run "./setup", you will obtain the following output:

```
==============================================================================

                           isCOBOL EVOLVE Installation
                           For isCOBOL Release 2021R1
                        Copyright (c) 2005 - 2021 Veryant

==============================================================================

Install Components:

    [0] All products..................................... (no)
    [1] isCOBOL Compiler (includes [2] & [3])............ (yes)
    [2] isCOBOL Runtime (includes [3])................... (no)
    [3] isCOBOL ThinClient............................... (no)
    [4] isCOBOL RemoteCompiler........................... (no)
    [5] isCOBOL EIS...................................... (no)
    [6] isCOBOL DatabaseBridge........................... (no)
    [7] isCOBOL Server................................... (no)
    [8] isCOBOL WebClient................................ (no)
    [9] isCOBOL LoadBalancer............................. (no)
   [10] isCOBOL Mobile................................... (no)

Install Path:
    [P] isCOBOL parent directory: UserHome

JDK Path:
    [J] JDK install directory: JavaHome

[S] Start Install        [Q] Quit

=============================================================================
Please press [ 1 2 3 4 5 6 7 8 P J S Q ]
```

8. Type "6", then press Enter to select isCOBOL DatabaseBridge.
9. (optional) Type "P", then press Enter to provide a custom installation path, if you don't want to keep the default one.
10. Type "S", then press Enter to start the installation.

    Note - if the setup script is not available for your Unix platform or you don't want to use it, just extract the tgz content to the folder where you want isCOBOL to be installed.

isCOBOL Evolve for UNIX/Linux provides shell scripts in the isCOBOL "bin" directory for compiling, running, and debugging programs. These scripts make use of two environment variables, ISCOBOL to locate the isCOBOL installation directory and ISCOBOL_JDK_ROOT to locate the JDK installation directory. To use these scripts set these environment variables and add the isCOBOL "bin" directory to your PATH.

For example, if you install isCOBOL in "/opt/isCOBOL" and your JDK is in "/opt/java/jdk1.8.0":

```
export ISCOBOL=/opt/isCOBOL
export ISCOBOL_JDK_ROOT=/opt/java/jdk1.8.0
export PATH=$ISCOBOL/bin:$PATH
```

Other Unix

A dedicated setup is provided for the following Unix platforms:

- Linux 32 bit
- Linux 64 bit
- FreeBSD
- Mac OSX 64 bit
- SunOS

If you need to install isCOBOL on another Unix platform, you can use the platform independent setup.

This setup includes only the cross platform items while it lacks native items. Contact Veryant if you need the porting of a native item to your Unix platform.

Instructions for the installation of the platform independent setup are provided below.

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .
2. Go to "https://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the "Platform Independent" section and select isCOBOL_2021_R1_*n*_noarch.tar.gz, where *n* is the build number.

Extract all contents of the archive:

```
gunzip isCOBOL_2021_R1_*_noarch.tar.gz
tar -xvf isCOBOL_2021_R1_*_noarch.tar
```

Distribution Files

For information on a specific distribution file, please see the README file installed with the product.

Activate the License

If you provided license keys during the installation, on Windows, you should skip reading this chapter.

isCOBOL DatabaseBridge looks for the following configuration property for license keys:

```
iscobol.easydb.license.2021=<license_key>
```

The keys should be stored in one of the following files (if they exist):

Windows

1. \etc\iscobol.properties in the drive where the working directory is
2. C:\Users\<username>\iscobol.properties (the setup wizard saves licenses here, if you don't skip activation)
3. iscobol.properties found in the Java Classpath
4. %ISCOBOL%\iscobol.properties
5. a custom configuration file passed on the command line

Unix/Linux

1. /etc/iscobol.properties
2. $HOME/iscobol.properties
3. iscobol.properties found in the Java Classpath
4. $ISCOBOL/iscobol.properties
5. a custom configuration file passed on the command line

NOTE - Files are listed in the order they're processed. If the license key appears in more than one of the above files, then the last occurrence is considered.

Working with DatabaseBridge

In order to have your COBOL programs work on a relational database instead of a standard ISAM file system, the following steps are necessary:

1. Generating EDBI user's routines
2. Compiling EDBI user's routines
3. Setting the isCOBOL environment
4. Configuring CLASSPATH and code_prefix

Generating EDBI user's routines

The creation of EDBI routines is made easy by the isCOBOL DatabaseBridge facility. With this feature enabled, every time the Compiler compiles a COBOL program with some indexed files defined in the File Section, it generates a bridge class that allows file operations to be reflected on a relational database.

The feature is activated and controlled through the Database Bridge (EasyDB) Configuration entries that can be set either in the configuration file or directly in the source through the SET Directive.

Let’s consider the COBOL program "PROG-FILE1.cbl" installed among isCOBOL samples. This program allows you to manage an archive of songs. It defines the following file:

```
       INPUT-OUTPUT         SECTION.
       file-control.
           select file1 assign to "file1"
           organization is indexed
           access mode is dynamic
           record key is f-cod
           file status is f-status.

       data division.
       file section.
       fd  file1.
       01  f-rec.
           03 f-cod       pic 9(3).
           03 f-firstname pic x(20).
           03 f-secname   pic x(20).
           03 f-address   pic x(20).
      $EFD DATE=YYYYMMDD
           03 f-birthday  pic 9(8).
```

There are two ways to generate EDBI routines:

- EDBI Generation at compile time (one step)
- EDBI Generation with EDBIIS (two steps)

EDBI Generation at compile time (one step)

EDBI routines can be generated automatically by the Compiler.

We can compile the program as follows:

```
iscc -c=compiler.properties PROG-FILE1.cbl
```

The file compiler.properties should include one or more DatabaseBridge configuration entries. For the moment, we just activate the database bridge feature. It will generate a generic EDBI routine. So, compiler.properties contains:

```
iscobol.compiler.easydb=1
```

At the end of the compilation process, you will find a folder named "easydb" with one file inside:

- genEDBI-file1.cbl : the bridge program that allows FILE1 to be used as a table on relational databases.

We’ve just demonstrated how to create a generic EDBI routine. Generic EDBI routines have some limitations, for example they don't manage locks and they can't check for table existence upon OPEN, so it's better to generate a more specific routine, depending on the RDBMS that we're going to use. With the next step, we’re going to generate a EDBI routine that is suitable for the Oracle database. To achieve it, change compiler.properties as follows:

```
iscobol.compiler.easydb=1
iscobol.compiler.easydb.oracle=1
```

Then compile again with the command:

```
iscc -c=compiler.properties PROG-FILE1.cbl
```

At the end of the compilation process, you will find a folder named "easydb" with one file inside:

- oraEDBI-file1.cbl : the bridge program that allows FILE1 to be used as a table on Oracle databases.

Refer to the installed sample README file for instructions about the deployment and testing of these items.

An alternative way to activate the DatabaseBridge facitlity is to set the necessary properties directly in the source code, using the SET Directive. For example:

```
      $set "easydb" "1"
      $set "easydb.oracle" "1"
       PROGRAM-ID. PROG-FILE1.
```

In this case, no specific configuration is required at compile time, so the compile command is just:

```
iscc PROG-FILE1.cbl
```

See Database Bridge (EasyDB) Configuration for the list of all the available compiler properties.

EDBI Generation with EDBIIS (two steps)

The EDBIIS command allows you to generate EDBI routines by processing XML data dictionaries.

In order to generate XML data dictionaries, compile the program as follows:

```
iscc -efd PROG-FILE1.cbl
```

At the end of the compilation process, you will find one additional file in your working directory:

- file1.xml : the data dictionary that describes FILE1.

Note - the way numeric fields are described in the dictionary is affected by the configuration property iscobol.compiler.efd_field_name_num (boolean).

Assuming that you want to generate a EDBI routine to use FILE1 on Oracle databases, you can run the following command:

```
edbiis -do file1.xml
```

It will generate:

- EDBI-file1.pco : the bridge program that allows FILE1 to be used as a table on Oracle databases.

See Command Line Options for the list of all the available edbiis options.

Compiling EDBI user's routines

If you relied on the automatic routine generation done by the Compiler, your EDBI routine will be automatically compiled using the same compiler options that were used for the original program.

If you generated the EDBI routines using EDBIIS, instead, then it's your duty to compile them. Take care to use the same data options that you are using with your COBOL application. For example, if you are compiling your programs with -ds for trailing separate sign, use -ds also to compile the EDBI routines.

Setting the isCOBOL environment

In this chapter we explain how to configure the runtime system in order to work on a Oracle database with isCOBOL DatabaseBridge.

Settings in iscobol.properties configuration file:

```
iscobol.file.index=easydb
iscobol.jdbc.driver=oracle.jdbc.OracleDriver
iscobol.jdbc.url=jdbc:oracle:thin:<<user>>/<<pwd>>@<<machine_name>>:<<port>>:<<sid>>
```

The last two properties refer to Oracle version 9i and may be different for other Oracle versions. Other databases have their own Driver and URL properties. Please refer to your database JDBC documentation and samples to see the correct values. You can also check the Data Access guide on JDBC for useful connection strings.

Configuring CLASSPATH and code_prefix

For the proper functioning of isCOBOL DatabaseBridge at run time, the following items must be available in the Classpath:

Classpath
• COBOL programs
• database JDBC driver jar library
• folder(s) with EDBI routine classes
• iscobol.jar library[A]

If iscobol.code_prefix is set in the configuration, then the above items must be distributed among Classpath and code_prefix as follows:

Classpath	iscobol.code_prefix
• database JDBC driver jar library	• COBOL programs
• iscobol.jar library[A]	• folder(s) with EDBI routine classes

[A] The isCOBOL wrappers take care of adding this item to the Classpath automatically.

The COBOL application is now ready to be run with the RDBMS.

EFD Directives

The generation of EDBI routines is based on the extended file description (EFD) generated by the Compiler.

If you generate the EDBI routines in one step at compile time, the EFD is kept in memory and you don't have to care about it.

 If you choose to generate the EDBI routines using EDBIIS, instead, you need to save the EFD to disc and this is achieved through the -efd compiler option.

Mapping rules

The EDBI routine will map only elementary fields to table columns and only the larger record type when there are multiple record definitions. Default rules used to map COBOL fields to table columns can be changed by using EFD directives.

Default Rules

COBOL file name	**RDBMS table**
COBOL record	table row
COBOL field	table column
COBOL key	table index

Redefines are allowed only for the entire record and must be redirected to a different RDBMS table (See WHEN Directive).

FILLER data items are not mapped to table columns. You can overwrite this behavior using the NAME Directive in order to associate a column name with COBOL FILLER.

Cobol group items are not mapped to table columns, instead they are mapped to elementary fields of the group. You can change this approach using the USE GROUP Directive.

There is not a corresponding RDBMS table definition for the COBOL OCCURS statement. For this reason the default behavior is to append a sequential number to the field name. For example:

```
01 myoccurs occurs 10 times.
    03 customer-code pic 9(5).
    03 customer-name pic x(30).
```

Will be mapped in RDBMS as

```
customer_code_1
customer-name_1
…
…
customer_code_3
customer-name_3
customer_code_4
customer-name_4
customer_code_5
customer-name_5
...
...
```

Using EFD directives

Directives are used when a COBOL file descriptor is mapped to a database field. The $EFD prefix indicates to the compiler that the proceeding command is used during the generation of the data dictionary.

Consult EFD Directives for a detailed description of the available EFD directives.

Invalid Data

Not all COBOL data is valid for the RDBMS.

DatabaseBridge transforms invalid data in the following way:

- if NULL is read from the database, they're stored as spaces in alphanumeric COBOL items and zero in numeric COBOL items.
- If numeric COBOL fields contain alphabetic digits, spaces, low-values or high-values, they're automatically converted according to the MOVE statement rules. Space is converted to zero, “A” to 1, and so on... If the -n option is used while parsing EFDs with edbiis, zero will be written when non-numeric data is encountered.
- If a COBOL field for which the DATE Directive was specified contains zero, spaces or low-values, the date specified by the iscobol.easydb.min_date configuration property is used.
- If a COBOL field for which the DATE Directive was specified contains high-values, the date specified by the iscobol.easydb.max_date configuation property is used.
- If a COBOL field for which the DATE Directive was specified contains an illegal date (i.e. 31th February), the date specified by the iscobol.easydb.inv_date configuration property is used.

Runtime Options and Configuration

In order to assign one or more indexed file to the DatabaseBridge at runtime, set the following configuration properties:

iscobol.file.index=easydb	Redirects all I/O to the appropriate EDBI routine
iscobol.file.index.*physicalfilename*=easydb	Redirects all I/O of *physicalfilename* to the appropriate EDBI routine. For example, setting iscobol.file.index.invoice=easydb means that the Dynamic Filesystem Interface will redirect all I/O done from any COBOL program that uses “invoice” as a physical file name (SELECT INVOICE ASSIGN TO “invoice”) to the EDBI-invoice routine.

iscobol.easydb.prefix=<prefix>	Instructs the runtime to call EDBI routine whose name begins with <prefix>. Possible values are the default values used at compile time: • db2 • d24 • gen • ifx • ora • mys • pgs • srv If you used a different prefix by setting *iscobol.compiler.easydb.<rdbms>.prefix* at compile time, use the same prefix here. **Note -** This property must not be set if the EDBI routines were generated by the EDBIIS command.
iscobol.jdbc.autocommit=<true|false>	This property should be set to true unless you wish to manage transactions or this documentation explicitly says to set it to false (e.g. when working on a MySQL or MariaDB database).

Configuration properties

The EDBI routines behavior is configurable through the following runtime properties.

The list of configuration properties that affect the Compiler behavior can be found at Database Bridge (EasyDB) Runtime Configuration.

Refer to the Configuration chapter for general information about setting configuration properties.

The EDBI-WHERE-CONSTRAINT external variable

Edbi-where-constraint is used to specify an additional WHERE condition for a succeeding START operation.

If you want to query city names that start with "A", add the following to your code:

```
WORKING-STORAGE SECTION.
01 edbi-where-constraint  pic x(300) is external.
...
PROCEDURE DIVISION.
...
*> to specify edbi-where-constraint

MOVE SPACES TO  edbi-where-constraint
OPEN I-O INVOICE
MOVE "city_name like 'A%'" to  edbi-where-constraint
MOVE SPACES TO INV-KEY
START INVOICE KEY IS NOT LESS INV-KEY
```

The A4GL-WHERE-CONSTRAINT external variable

The EDBI routines work in Acucobol-GT compatibility in these two cases

- if they are generated by the EDBIIS command and the -ca option is used on the EDBIIS command line, or
- if they are generated by the isCOBOL Compiler and the COBOL program is compiled with -ca option and iscobol.compiler.easydb (boolean) set to true.

When the Acucobol-GT compatibility is active, the A4GL-WHERE-CONSTRAINT external variable is evaluated instead of EDBI-WHERE-CONSTRAINT. The definition and usage of A4GL-WHERE-CONSTRAINT are the same as EDBI-WHERE-CONSTRAINT. The only difference between these two external data items is that A4GL-WHERE-CONSTRAINT is automatically initialized before the next START statement.

If you want to query city names that start with "A" in Acucobol-GT compatibility, add the following to your code:

```
WORKING-STORAGE SECTION.
01 a4gl-where-constraint  pic x(300) is external.
...
PROCEDURE DIVISION.
...
*> to specify edbi-where-constraint

MOVE SPACES TO  edbi-where-constraint
OPEN I-O INVOICE
MOVE "city_name like 'A%'" to  a4gl-where-constraint
MOVE SPACES TO INV-KEY
START INVOICE KEY IS NOT LESS INV-KEY
```

iscobol.easydb.limit_dropdown

When a sequence of START and READ NEXT/PREVIOUS operations are performed by an application, EDBI routines generate a sequence of queries to return the set of records matching the application's request. To improve performance, the interface generates a sequence of DROP DOWN queries based upon the key of reference's key segments going from the most specific subset using the most number of segments to the most general using the least number of segments.

For example if a key is described by:

```
       03 FILE1-KEY.
          05 FILE1-KEY-SEG1      PIC X(2).
          05 FILE1-KEY-SEG2      PIC 9(9).
          05 FILE1-KEY-SEG3      PIC X.
```

then a START followed by a sequence of READ NEXT operations might generate the selection criteria of:

1. WHERE FILE1-KEY-SEG1 = :w1 AND FILE1-KEY-SEG2 = :w2 AND FILE1-KEY-SEG3 >= :w3
2. WHERE FILE1-KEY-SEG1 = :w1 AND FILE1-KEY-SEG2 > :w2
3. WHERE FILE1-KEY-SEG1 > :w1

This can improve performance in that the target for each query is kept to a minimal size. If a set of records is not required, the database does not need to spend the time building the working set. When a DROP DOWN does occur however, the subsequent working set can require a large amount of time to process, because it may be an order of magnitude greater number of records. Normally there is not a way for a COBOL application to instruct the interface to stop processing when it has finished with the records based on a given key segment.

The iscobol.easydb.limit_dropdown configuration property provides this instruction. It allows an application to direct the interface not to perform DROP DOWN query generation and instead return end of file when the records matching the current query have been exhausted. When this property is set to 1, if the record positioning was performed by a START with a SIZE clause, such that the initial positioning was performed using fewer than the total number of columns in the key, the process will cease after all records matching the START columns have been exhausted.

DatabaseBridge generator (edbiis)

The EDBIIS command allows the user to generate EDBI routines for supported RDBMS.

This command is mainly supported for backward compatibility. If you activated the DatabaseBridge as described in EDBI Generation at compile time (one step), then you don't need to use this command.

Instead, if you wish to generate EDBI routines in a separate step by processing EFD dictionaries, then you need this command.

Syntax

```
edbiis -help|[options] <efdfilename>
```

Command Line Options

The table below lists the available command line options. The name of the corresponding Compiler configuration property for each option is provided.

The compiler configuration is considered only if iscobol.compiler.easydb (boolean) is set to true.

Command line option	Corresponding Compiler configuration	Description
(default)	(default)	Generates EDBI routines for generic RDBMS.
-d2	iscobol.compiler.easydb.db2=1	Generates EDBI routines for DB2 RDBMS.
-d4	iscobol.compiler.easydb.db2_as400=1	Generates EDBI routines for DB2/AS400 RDBMS.
-di	iscobol.compiler.easydb.informix=1	Generates EDBI routines for Informix (certified for ANSI-mode databases).
-do	iscobol.compiler.easydb.oracle=1	Generates EDBI routines for ORACLE RDBMS.
-dm	iscobol.compiler.easydb.mysql=1	Generates EDBI routines for MySQL (InnoDB engine) and MariaDB.

Command line option	Corresponding Compiler configuration	Description

-dmld	iscobol.compiler.easydb.mysql=1
	iscobol.compiler.easydb.light_cursors=2
		Generates EDBI routines with light cursors for MySQL (InnoDB engine) and MariaDB. See iscobol.easydb.mysql_row_limit for details.
		
		Note - An additional column named OID is generated in the tables when this option is used. For this reason, routines generated with this option can't work on tables that were created by routines generated with -dm or -dmlu options and vice versa.

-dmlu	iscobol.compiler.easydb.mysql=1
	iscobol.compiler.easydb.light_cursors=1
		Generates EDBI routines with light cursors for unique indexes for MySQL (InnoDB engine) and MariaDB. See iscobol.easydb.mysql_row_limit for details.

-dmoid=<name>	iscobol.compiler.easydb.mysql.oid_name =<name>	Specifies the name of the OID field generated when -dmld is used. The default name is "OID".
-dp	iscobol.compiler.easydb.postgres=1	Generates EDBI routines for PostgreSQL.

-dpld	iscobol.compiler.easydb.postgres=1
	iscobol.compiler.easydb.light_cursors=2
		Generates EDBI routines with light cursors for PostgreSQL. See iscobol.easydb.postgres_row_limit for details.

-dplu	iscobol.compiler.easydb.postgres=1
	iscobol.compiler.easydb.light_cursors=1
		Generates EDBI routines with light cursors for unique indexes for PostgreSQL. See iscobol.easydb.postgres_row_limit for details.

-dpoid=<name>	iscobol.compiler.easydb.postgres.oid_name=<name>	Specifies the name of the OID field generated when -dpld is used. The default name is "OID".
-ds	iscobol.compiler.easydb.sqlserver=1	Generates EDBI routines for Microsoft SQL Server.
For every database:		
-ca	(inherited from the Compiler command line)	Uses A4GL-WHERE-CONSTAINT instead of The EDBI-WHERE-CONSTRAINT external variable and implies -defCHAR.

Command line option	Corresponding Compiler configuration	Description
-cc	(default)	Includes the COMMIT COUNT feature in the EDBI routine. A COMMIT statement is automatically performed after a given number of successful WRITE, REWRITE and DELETE operations. The number is configured by the properties iscobol.easydb.commit_count.ConnectionName and iscobol.easydb.commit_count. Note that the COMMIT COUNT feature requires iscobol.jdbc.autocommit (boolean) to be set to false in the configuration, otherwise all the operations are automatically committed.
-ce	(default)	Generate direct SELECT query instead of using a CURSOR to perform READ KEY.
-csqq	(inherited from the Compiler command line)	Generates identifiers between quotes. This option is useful in two situations: 1) if you need to keep identifiers case sensitive 2) if you have fields whose name starts with a number In the second scenario, you should also set iscobol.compiler.efd_field_name_num (boolean) to true in the Compiler configuration when you compile the program that includes the FD of the table.
-defCHAR	iscobol.compiler.easydb.defchar=1	Manages alphanumeric fields using CHAR instead of VARCHAR. Use the VAR-LENGTH Directive to mark specific alphanumeric fields as VARCHAR.
-entrypoints	iscobol.compiler.easydb.entry_points=1	Generates entry points where the user can inject customized code. See Extending EDBI routines through entry points for more information.
-esst	(default)	Generates additional code to provide the ability to use only the table related to a specific record type in multi-record files. To enable the feature at runtime, set iscobol.easydb.start_on_specific_table (boolean) to true in the configuration.

Command line option	Corresponding Compiler configuration	Description
-h	iscobol.compiler.easydb.high_values_as_max_val=1	Replaces HIGH-VALUE with the maximum numeric value in numeric fields. This option affects numeric items that cannot be set to HIGH-VALUE. It doesn't affect COMP, BINARY, COMP-X, COMP-5 and COMP-2 as well as numeric items for which either the ALPHA Directive or the DATE Directive were used.
-i	iscobol.compiler.easydb.isam_eof=1	ISAM positioning on at end Using the -i option produces a different behavior when reversing direction after reading past the beginning or end of a file. The record returned by the READ PREVIOUS is the second-to-last record in the file, and the record returned by the READ NEXT is the second record in the file.
-jcd=*<routine_name>*	iscobol.compiler.easydb.julian_routines=*<routine_name>;<routine_name>*	Specifies an alternate routine for the conversion of julian dates before writing on the database. By default EDBI_DTJUCBDB (installed with the product) is used. This routine takes advantage of the DATE-OF-INTEGER intrinsic function for the conversion. If you wish to write your own routine that uses a different conversion logic, use the same Linkage parameters as EDBI_DTJUCBDB.CBL found in easydb\edbisource in the isCOBOL installation. The custom routine you provide is searched in the iscobol.code_prefix, if set, or in the Class Path otherwise. The -jcd option must be used in conjunction with -jdc.

Command line option	Corresponding Compiler configuration	Description
-jdc=<*routine_name*>	iscobol.compiler.easydb.julian_routines=<*routine_name*>;<*routine_name*>	Specifies an alternate routine for the conversion of julian dates after reading from the database. By default EDBI_DTJUDBCB (installed with the product) is used. This routine takes advantage of the INTEGER-OF-DATE intrinsic function for the conversion. If you wish to write your own routine that uses a different conversion logic, use the same Linkage parameters as EDBI_DTJUDBCB.CBL found in easydb\edbisource in the isCOBOL installation. The custom routine you provide is searched in the iscobol.code_prefix, if set, or in the Class Path otherwise. The -jdc option must be used in conjunction with -jcd.
-maxCHARlen=*n*	iscobol.compiler.easydb.max_char_len==*n*	Alphanumeric fields whose size is not greater than *n* are managed as CHAR, the others are managed as VARCHAR. This option is not compatible with -defCHAR and overrides both FIX-LENGTH Directive and VAR-LENGTH Directive.
-mo	(not available)	Generates multitable subroutines using standard COBOL statements instead of object oriented syntax.
-n	iscobol.compiler.easydb.test_not_numeric=1	Test not numeric Using the -n option will include an additional test on numeric key fields to verify whether a numeric value is used. This additional check will determine if non-numeric values are used and replace those non-numeric values with 0.

Command line option	Corresponding Compiler configuration	Description

-no	iscobol.compiler.easydb.names_with_leading_zeros=1	Use leading zeroes in OCCURS item names. The number of leading zeroes depends by the occurs size. EasyDB puts before as many zeroes as it takes to reach the number of digits of the occurs size.
		
		Example:
		Consider the following COBOL items:
		`  03 my_item_a pic x(10) occurs 3.`
		`  03 my_item_b pic x(10) occurs 30.`
		`  03 my_item_c pic x(10) occurs 300.`
		Without -no option the columns generated by edbiis are named:
		`  my_item_a_1, my_item_a_2, my_item_a_3`
		`  my_item_b_1, my_item_b_2, my_item_b_3, ... my_item_b_30`
		`  my_item_c_1, my_item_c_2, my_item_c_3, ... my_item_c_300`
		With -no option the columns are named:
		`  my_item_a_1, my_item_a_2, my_item_a_3`
		`  my_item_b_01, my_item_b_02, my_item_b_03, ... my_item_b_30`
		`  my_item_c_001, my_item_c_002, my_item_c_003, ... my_item_c_300`

-nocheck	iscobol.compiler.easydb.no_check=1	Avoid checking for table existence during the OPEN statement. We assume that the table exists. The file not found error is never returned by the OPEN statement and the optional files are not automatically created if they don't exist.
-od=<*dirname*>	iscobol.compiler.easydb.output=<*dirname*>	Output directory for EDBI routines.
-od=<*dirname*>	iscobol.compiler.easydb.sql.output=<*dirname*>	Output directory for sql files.
-pdo	iscobol.compiler.easydb.duplicates_in_order=1	List records with duplicate keys values ordered by the primary key during READ NEXT and READ PREVIOUS on alternate keys with duplicates. Without this option there is no guarantee to read the correct record by reading on the opposite way when the file pointer is on a duplicated key value. This option might slow down performance.

Command line option	Corresponding Compiler configuration	Description
-sl	(default)	Support for START WITH SIZE. Using this option, edbiis stores additional code in the routine to handle the SIZE clause of the START statement. If the -sl option is omitted, the routine will handle the START statement as if the SIZE clause is not specified.
-sql	iscobol.compiler.easydb.sql=1	Generates a script file with .sql extension that includes the CREATE TABLE statement.
-t	iscobol.compiler.easydb.test_not_numeric=2	Allow trace for not numeric The -t option must be used along with -n. Routines generated with -t and -n options keep trace of cases of *not numeric data in numeric field* in a separate log file whose name is controlled by the iscobol.edbi.notnum.tracefile configuration property.
-ua	iscobol.compiler.easydb.unlock_all=1	Generate additional code to provide support for the statement UNLOCK file-name ALL RECORDS. The statement UNLOCK ALL is always ignored, instead. Warning: In order to unlock the file, the EDBI routine will issue a COMMIT statement, so every active lock will be lost.
-v	(inherited from the Compiler command line)	Show product version.
Only for Informix:		
-Id	(not available)	Use strings to represent date values in SQL statements. Avoid conversion functions. This is useful when working with old Informix versions where date conversion functions were not available.
Only for Oracle:		
-oh	iscobol.compiler.easydb.oracle.hints=1	Generate Oracle optimizer hints that force the query optimizer to use the proper index. This option is deprecated and supported only for backward compatibility. Unless the HINT Directive has been used to specify custom hints, you may consider using the new option -oho.

Command line option	Corresponding Compiler configuration	Description
-oho	iscobol.compiler.easydb.oracle.hints=2	Generate Oracle optimizer hints that force the query optimizer to use the proper index. Hints are also used to specify the data ordering, avoiding the Order By clause and providing better performance. The HINT Directive shouldn't be used along with this option since it might cause wrong data ordering. This option is incompatible with the -Oh option and with the configuration setting iscobol.jdbc.cursor.type=3.
-oii=<*integer*>	iscobol.compiler.easydb.oracle.index_storage_initial_value=<*integer*>	Initial storage value for index.
-oit=<*integer*>	iscobol.compiler.easydb.oracle.table_storage_initial_value=<*integer*>	Initial storage value for table.
-oni=<*integer*>	iscobol.compiler.easydb.oracle.index_storage_next_value=<*integer*>	Next storage value for index.
-ont=<*integer*>	iscobol.compiler.easydb.oracle.table_storage_next_value=<*integer*>	Next storage value for table.
-opi=<*integer*>	iscobol.compiler.easydb.oracle.index_storage_pctincrease_value=<*integer*>	pctincrease storage value for index.
-opt=<*integer*>	iscobol.compiler.easydb.oracle.table_storage_pctincrease_value=<*integer*>	pctincrease storage value for table.
-oti=<*name*>	iscobol.compiler.easydb.oracle.tablespace_index_name=<*name*>	Tablespace index name.
-ott=<*name*>	iscobol.compiler.easydb.oracle.tablespace_name=<*name*>	Tablespace name.
-ow	iscobol.compiler.easydb.oracle.lock_wait=0	NOWAIT for update. This option allows the EDBI user's routine to return record lock condition.
-owfl	iscobol.compiler.easydb.oracle.wait_for_locks=1	Includes the WAIT FOR LOCKS feature in the EDBI routine. Before each READ operation the EDBI routine tests the iscobol.easydb.wait_for_lock (boolean) configuration property. If the property is set to true, then the lock condition is not returned and the program waits for the lock to be released. If the property is set to false, then the lock condition is returned. This option can't be used along with -Ow.

Command line option	Corresponding Compiler configuration	Description
Only for Ms SQL Server:		
-sc	iscobol.compiler.easydb.sqlserver.latin1_general_bin=1	Use standard ASCII comparison when sorting data. By default, without this option, in collation comparisons that use Windows collations, characters like a single quote (') or hyphen (-) are compared last, only after the regular alphabet characters are compared. With this option *COLLATE Latin1_General_Bin* is used upon CREATE TABLE.
-sco	iscobol.compiler.easydb.sqlserver.latin1_general_bin=2	Use standard ASCII comparison when sorting data. By default, without this option, in collation comparisons that use Windows collations, characters like a single quote (') or hyphen (-) are compared last, only after the regular alphabet characters are compared. With this option *COLLATE Latin1_General_Bin* is used in the ORDER BY clause of queries.
-sdt	iscobol.compiler.easydb.sqlserver.datetime_always=1	Use always DATETIME to represent COBOL fields with the EFD DATE directive, regardless of the date format string.
Only for MySQL and MariaDB:		
-mh	iscobol.compiler.easydb.mysql.hints=true	Generate MySQL optimizer hints that force the query optimizer to use the proper index.
Only for PostgreSQL:		
-pi	(default)	Use indicator variables to manage COBOL Low-Values as NULL on the database.

Unlike other drivers, the PostgreSQL JDBC driver tries to load all the records of a Cursor (object used by EDBI subroutines to store table records) into memory. For this reason, when the program performs a START on a huge table, an out of memory error may occur. The -dpld and -dplu options help to avoid this situation. When used, the EDBI subroutine will include a pagination logic that keeps the Cursor light. Use one of these options instead of -dp if you plan to work on huge tables with PortgreSQL.

Processing multiple EFD files at once

The EDBIIS command supports the * wildcard in the *efdfilename* parameter.

For example, the following operations:

```
edbiis file1.xml
edbiis file2.xml
edbiis file3.xml
edbiis file4.xml
```

can be done all at once with the command:

```
edbiis file*.xml
```

EDBI Routines

In this chapter we describe the standard EDBI routines installed along with the runtime system and the EDBI routines that can be generated for every supported database.

EDBI Standard Routines

EDBI standard routines are included in the runtime library (iscobol.jar).

You will find the *edbisource* directory, where all of the source code of internal EDBI routines under the *easydb* directory on a standard isCOBOL root installation. You can customize their code and rely on the compile scripts stored in the same directory to create a customized library whose classes will be used as replacement of the default ones.

Three of the routines, EDBI-COMMIT.cbl, EDBI-ROLLBACK.cbl and EDBI-CONNECT.cbl allow users to adapt SQL during COMMIT, ROLLBACK and CONNECT step.

Twelve of the COBOL routines are for data conversion, EDBI-DT6DCBDB.cbl, EDBI-DT6DDBCB.cbl, EDBI-DT6MCBDB.cbl, EDBI-DT6MDBCB.cbl, EDBI-DT6YCBDB.cbl, EDBI-DT6YDBCB.cbl, EDBI-DT8DCBDB.cbl, EDBI-DT8DDBCB.cbl, EDBI-DT8MCBDB.cbl, EDBI-DT8MDBCB.cbl, EDBI-DT8YCBDB.cbl, EDBI-DT8YDBCB.cbl.

If you wish to customize one or more of these routines, proceed as follows:

1. edit the source code of the EDBI routine that you wish to customize
2. compile the routine with -sysc option
3. put the resulting class file in a folder or a jar library
4. add the folder or the jar to the beginning of the Classpath setting

EDBI Routines for Generic RDBMS

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	CHAR(n)
PIC 9(n)	NUMERIC(n)

PIC 9(n)V9(m)	NUMERIC(n+m,m)
PIC S9(n)	NUMERIC(n)
PIC S9(n)V9(m)	NUMERIC(n+m,m)

Limitations:

No record lock support.

No check for table existence (OPEN INPUT / I-O)

Peculiar jdbc settings:

None.

EDBI Routines for Generic DB2 RDBMS

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	VARCHAR(n)
PIC 9(n)	DECIMAL(n)
PIC 9(n)V9(m)	DECIMAL(n+m,m)
PIC S9(n)	DECIMAL(n)
PIC S9(n)V9(m)	DECIMAL(n+m,m)

Peculiar jdbc settings:

None.

EDBI Routines for DB2/400

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	VARCHAR(n)
PIC 9(n)	DECIMAL(n)
PIC 9(n)V9(m)	DECIMAL(n+m,m)
PIC S9(n)	DECIMAL(n)
PIC S9(n)V9(m)	DECIMAL(n+m,m)

Peculiar jdbc settings:

None.

EDBI Routines for ORACLE RDBMS

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	VARCHAR2(n)
PIC 9(n)	NUMERIC(n)
PIC 9(n)V9(m)	NUMERIC(n+m,m)
PIC S9(n)	NUMERIC(n)
PIC S9(n)V9(m)	NUMERIC(n+m,m)

Peculiar jdbc settings:

None.

EDBI Routines for MySQL (InnoDB engine) and MariaDB

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	VARCHAR(n)
PIC 9(1-2)	TINYINT
PIC 9(3-4)	SMALLINT
PIC 9(5-6)	MEDIUMINT
PIC 9(7-9)	INT
PIC 9(>9)	BIGINT
PIC 9(n)V9(m)	DECIMAL(n+m,m)
PIC S9(n)V9(m)	DECIMAL(n+m,m)

Peculiar jdbc settings:

`iscobol.jdbc.autocommit=false`	This is set in order to take a lock if issued. The COBOL program shouldn't use COMMIT and ROLLBACK statements in order to avoid conflicts with the operations performed by EDBI routines.
`iscobol.easydb.commit_count=1`	This is set in conjunction with *iscobol.jdbc.autocommit=false* in order to update the table at each WRITE, REWRITE and DELETE statement. Otherwise updates would be made only at CLOSE.
`iscobol.jdbc.on_stop_run=commit`	Due to the above setting, it's good practice to instruct the runtime to commit all modifications before exiting.

Lock Timeout:

By default the MySQL and MariaDB drivers wait for locks to be released. If you wish to receive a 'record locked' error, you need to issue this statement after the connection has been acquired (e.g. after the opening of the first file):

```
           EXEC SQL
                EXECUTE IMMEDIATE
                "set innodb_lock_wait_timeout = 0"
           END-EXEC
```

Note - the above peculiar settings and the need of setting the lock wait timeout are required only if you wish to manage locks natively on the database. If you're working in a Application Server (Thin Client) or File Server environment and you wish to have a full support for locking features, then you may consider handling locks through the Internal lock management.

EDBI Routines for Microsoft SQL Server

Data mapping (any COMP type could be used, mapping is done according to the digits):

PIC X(n)	VARCHAR(n)
PIC 9(1-4)	SMALLINT
PIC 9(5-9)	INT
PIC 9(>9)	BIGINT
PIC 9(n)V9(m)	DECIMAL(n+m,m)
PIC S9(n)V9(m)	DECIMAL(n+m,m)

Peculiar jdbc settings:

```
iscobol.jdbc.cursor.concurrency = 1009
```

Lock Timeout:

By default the SQL Server JDBC driver waits for locks to be released. If you wish to receive a 'record locked' error, you can configure the lock timeout in the connection URL, for example:

```
iscobol.jdbc.url=jdbc:sqlserver://my-
server:1433;user=sa;password=manager;lockTimeout=1000
```

Locking Tables:

The statements OPEN EXCLUSIVE I-O and OPEN I-O WITH LOCK have no effect on empty tables. In order to acquire a lock on the whole file, at least one record must be present.

Note - the above peculiar settings and the need of setting the lock timeout are required only if you wish to manage locks natively on the database. If you're working in a Application Server (Thin Client) or File Server environment and you wish to have a full support for locking features (including OPEN EXCLUSIVE and OPEN I-O WITH LOCK), then you may consider handling locks through the Internal lock management.

EDBI Routines for PostgreSQL

Data mapping (any COMP type could be used, mapping it is done according with digits):

PIC X(n)	VARCHAR(n)
PIC 9(n)	NUMERIC(n)
PIC 9(n)V9(m)	NUMERIC(n+m,m)
PIC S9(n)V9(m)	NUMERIC(n+m,m)

Peculiar jdbc settings:

None.

EDBI Routines for Informix

Data mapping (any COMP type could be used, mapping it is done according with digits):

PIC X(n)	VARCHAR(n)
PIC 9(n)	NUMERIC(n)
PIC 9(n)V9(m)	NUMERIC(n+m,m)
PIC S9(n)V9(m)	NUMERIC(n+m,m)

Peculiar jdbc settings:

None.

Extending EDBI routines through entry points

The code of EDBI routines can be customized by adding additional operations in dedicated entry points.

This feature is activated by the iscobol.compiler.easydb.entry_points (boolean) Compiler's configuration property or by the -entrypoints option of EDBIIS.

If the feature is activated, the generated EDBI routine will reference the following copybooks:

Copybook	**Content**
edbi.ini	The ENVIRONMENT DIVISION of the program. Here you can specify SPECIAL-NAMES as well as a REPOSITORY for classes that you wish to reference.
edbi.wrk	Additional WORKING-STORAGE data items.

Copybook	Content
edbi.prd	Additional PROCEDURE DIVISION code. The following paragraphs must be included here: BEFORE-TABLE-OPEN. BEFORE-TABLE-OPEN-EX. AFTER-TABLE-OPEN. AFTER-TABLE-OPEN-EX. BEFORE-TABLE-CLOSE. BEFORE-TABLE-CLOSE-EX. AFTER-TABLE-CLOSE. AFTER-TABLE-CLOSE-EX. BEFORE-TABLE-INSERT. BEFORE-TABLE-INSERT-EX. AFTER-TABLE-INSERT. AFTER-TABLE-INSERT-EX. BEFORE-TABLE-UPDATE. BEFORE-TABLE-UPDATE-EX. AFTER-TABLE-UPDATE. AFTER-TABLE-UPDATE-EX. BEFORE-TABLE-DELETE. BEFORE-TABLE-DELETE-EX. AFTER-TABLE-DELETE. AFTER-TABLE-DELETE-EX. BEFORE-TABLE-READ. BEFORE-TABLE-READ-EX. BEFORE-TABLE-LOCK. AFTER-TABLE-LOCK. AFTER-TABLE-LOCK-EX. BEFORE-TABLE-UNLOCK. BEFORE-TABLE-UNLOCK-EX. AFTER-TABLE-UNLOCK. AFTER-TABLE-UNLOCK-EX. BEFORE-DROP-CREATE. BEFORE-DROP-CREATE-EX. AFTER-DROP-CREATE. AFTER-DROP-CREATE-EX. BEFORE-TABLE-DROP. BEFORE-TABLE-DROP-EX. AFTER-TABLE-DROP. AFTER-TABLE-DROP-EX. BEFORE-TABLE-START. BEFORE-TABLE-START-EX. AFTER-TABLE-START. AFTER-TABLE-START-EX.

It’s your duty to create these copybooks and make them available at compile time.

The copybooks host the data items and the statements that you wish to add to the standard EDBI routine code. BEFORE-operation and AFTER-operation paragraphs are performed by the EDBI routine before and after each i-o operation.

Working with multiple connections

By default, DatabaseBridge works on the connection identified by the iscobol.jdbc.driver and iscobol.jdbc.url configuration properties, however it's possible to define multiple connections and associate the COBOL files with them.

In order to define multiple connections in the isCOBOL configuration, the following syntax must be used:

```
iscobol.jdbc.driver.<connection_name>
iscobol.jdbc.url.<connection_name>
iscobol.easydb.prefix.<connection_name>
```

Note - *iscobol.easydb.prefix* should be set only if you generated the subroutines as described in EDBI Generation at compile time (one step).

The above syntax must be repeated for each connection you wish to define varying *connection_name*. Different connections can be

- on the same database
- on different databases of the same family (e.g. Oracle)
- on different databases of different families (e.g. Oracle and MySQL)

Once there are some connections defined in the configuration, you can associate single files with them with the following syntax:

```
iscobol.easydb.connection_name.<physical_file_name>=<connection_name>
```

Files that are not explicitly associated with a connection will use the default connection identified by the iscobol.jdbc.driver and iscobol.jdbc.url settings.

Example:

```
iscobol.jdbc.driver=com.mysql.jdbc.Driver
iscobol.jdbc.url=jdbc.mysql://localhost:3306/msqldb?user=scott&password=tiger
iscobol.easydb.prefix=mys

iscobol.jdbc.driver.conn_ora=oracle.jdbc.OracleDriver
iscobol.jdbc.url.conn_ora=jdbc:oracle:thin:scott/tiger@localhost:1521:orcl
iscobol.easydb.prefix.conn_ora=ora

iscobol.jdbc.driver.conn_post=org.postgresql.Driver
iscobol.jdbc.url.conn_post=jdbc:postgresql:pgdb:scott/tiger@localhost:1522:
iscobol.easydb.prefix.conn_post=pgs

iscobol.easydb.connection_name.file1=conn_post
iscobol.easydb.connection_name.file2=conn_ora
```

Using the above settings, FILE1 is handled on PostgreSQL and FILE2 is handled on Oracle, while any other file is handled on MySQL.

Managing multi-record files

The COBOL language allows multiple records to be described in the same FD. These records appear as different level 01 items redefining each other.

DatabaseBridge allows you to manage this situation by redirecting each record definition to a different table.

Consider the following FD:

```
       FD  filem.
       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).
       01  f-recE.
           03 f-keyE.
              05 f-codE        pic 9(3).
              05 f-typeE       pic x.
           03 italian-person.
              05 E-nome        pic x(32).
              05 E-cognome     pic x(32).
              05 E-indirizzo   pic x(32).
              05 E-cap         pic x(5).
              05 new-field-A   pic x(5).
              05 filler        pic x(15).
       01  f-recS.
           03 f-keyS.
              05 f-codS         pic 9(3).
              05 f-typeS        pic x.
           03 ASIAN-FIELD       pic x(121).
       01  f-recF.
           03 f-keyF.
              05 f-codF         pic 9(3).
              05 f-typeF        pic x.
           03 AFRICAN-FIELD1    pic x(120).
           03 AFRICAN-FIELD2    pic x(1).
       01  f-recU.
           03 f-keyU.
              05 f-codU         pic 9(3).
              05 f-typeU        pic x.
           03 AUSTRALIAN-FIELD1    pic x(40).
           03 AUSTRALIAN-FIELD2    pic x(40).
           03 AUSTRALIAN-FIELD3    pic x(40).
           03 AUSTRALIAN-FIELD4    pic x(1).
```

By default, only the first record definition would be handled on the database. This behavior is transparent for the COBOL program because the runtime automatically sets redefined fields, but it doesn't allow all of the fields on the database to be seen, so other client tools external to the COBOL program cannot manage them.

In order to bring all the fields on the database, it's necessary to specify a condition and a destination table for every record definition thru the EFD WHEN Directive. The above FD can be changed to:

```
        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 italian-person.
               05 E-nome        pic x(32).
               05 E-cognome     pic x(32).
               05 E-indirizzo   pic x(32).
               05 E-cap         pic x(5).
               05 new-field-A   pic x(5).
               05 filler        pic x(15).
       $EFD WHEN F_TYPE = "S" TABLENAME = ASIAN_PEOPLE
        01  f-recS.
            03 f-keyS.
               05 f-codS         pic 9(3).
               05 f-typeS        pic x.
            03 ASIAN-FIELD       pic x(121).
       $EFD WHEN F_TYPE = "F" TABLENAME = AFRICAN_PEOPLE
        01  f-recF.
            03 f-keyF.
               05 f-codF         pic 9(3).
               05 f-typeF        pic x.
            03 AFRICAN-FIELD1    pic x(120).
            03 AFRICAN-FIELD2    pic x(1).
       $EFD WHEN F_TYPE = "U" TABLENAME = AUSTRALIAN_PEOPLE
        01  f-recU.
            03 f-keyU.
               05 f-codU         pic 9(3).
               05 f-typeU        pic x.
            03 AUSTRALIAN-FIELD1    pic x(40).
            03 AUSTRALIAN-FIELD2    pic x(40).
            03 AUSTRALIAN-FIELD3    pic x(40).
            03 AUSTRALIAN-FIELD4    pic x(1).
```

This causes multiple EDBI routines to be generated.

The runtime will automatically call the proper EDBI routine depending on the condition validated on the current record.

Managing relative and sequential files on the database

DatabaseBridge allows to manage relative and sequential files as database tables.

The table generated for a relative or sequential file includes two additional fields:

RELSEQ_DUMMY_KEY	A numeric column that stores the ordinal number of the record. This field is used to read the records in the order they were written, like it happens with disk files. This is the primary key of the table.
RELSEQ_DUMMY_RLEN	A numeric column that stores the record length of the current record. The value may change from record to record if the file has a variable length record.

EDBI Generation at compile time (one step)

EDBI routines can be generated automatically by the Compiler.

We can compile programs as follows:

```
iscc -c=compiler.properties <program_name>.cbl
```

The file compiler.properties should include one or more DatabaseBridge configuration entries.

In order to activate the support for relative and sequential files, at least these two entries must be included:

```
iscobol.compiler.easydb=1
iscobol.compiler.easydb.index_only=0
```

At the end of the compilation process, you will find additional source files in your working directory:

- <prefix>EDBI-<filename>.cbl : the bridge program that allows the file identified by filename to be used as a table on relational databases. The prefix depends by the destination database; for example, if *iscobol.compiler.easydb.oracle=1* is used, then oraEDBI-<filename>.cbl is generated.

An alternative way to activate the DatabaseBridge facitlity is to set the necessary properties directly in the source code, using the SET Directive. For example:

```
      $set "easydb" "1"
      $set "easydb.oracle" "1"
      $set "easydb.index_only" "0"
       PROGRAM-ID. PROG-FILE1.
```

In this case, no specific configuration is required at compile time, so the compile command is just:

```
iscc <program_name>.cbl
```

Note - the setting *easydb.light_cursors* is not supported for relative and sequential files.

EDBI Generation with EDBIIS (two steps)

The EDBIIS command allows you to generate EDBI routines by processing XML data dictionaries.

In order to generate XML data dictionaries for relative and sequential files, compile the program as follows:

```
iscc -efa <program_name>.cbl
```

At the end of the compilation process, you will find additional files in your working directory:

- <filename>.xml : the data dictionary that describes the file identified by filename.

In order to generate EDBI routines, use the following command:

```
edbiis <options> <filename>.xml
```

For example, in order to generate EDBI routines for the Oracle database, you can run:

```
edbiis -do <filename>.xml
```

It will generate:

- EDBI-<filename>.pco : the bridge program that allows the file identified by filename to be used as a table on Oracle databases.

Note - the options *-dmld*, *-dmlu*, *-dpld* and *-dplu* are not supported for relative and sequential files.

Compiling EDBI user's routines

If you relied on the automatic routine generation done by the Compiler, your EDBI routine will be automatically compiled using the same compiler options that were used for the original program.

If you generated the EDBI routines using EDBIIS, instead, then it’s your duty to compile them. Take care to use the same data options that you are using with your COBOL application. For example, if you are compiling your programs with -ds for trailing separate sign, use -ds also to compile the EDBI routines.

Setting the isCOBOL environment

In order to have relative and sequential files managed as database tables, add the following entries to the runtime configuration:

```
iscobol.file.linesequential=easydb
iscobol.file.relative=easydb
iscobol.file.sequential=easydb
```

Ensure that iscobol.jdbc.driver and iscobol.jdbc.url are also set in order to allow the connection to the database.

Differences between DatabaseBridge and ISAM

DatabaseBridge allows you to keep the same approach used for ISAM files despite it works on a relational database. However there are few differences between the standard ISAM and the ISAM simulated by DatabaseBridge. These differences are discussed below.

Locks Management

To emulate the COBOL locks behavior, DatabaseBridge takes advantages of specific database features that are different from database to database.

The lock of a single record is fully supported when using Oracle, Informix, MS SQL Server, MySQL, MariaDB, DB2 and PostgreSQL.

The lock on multiple records is not officially supported. DatabaseBridge always takes one lock at a time because it re-uses the same ESQL cursor for reading a record with lock. If the program locks multiple records working, it depends by a database configuration that keeps locks alive.

The lock a whole file/table is supported when using Oracle, Informix, MS SQL Server, DB2 and PostgreSQL. MySQL and MariaDB don't support the ability to lock a whole file/table.

The following operations are not currently supported by the DatabaseBridge. Contact Veryant if you need any of these capabilities:

- Lock between two programs in the same runtime session
- Unlock all records of all open files with UNLOCK ALL
- Unlock a record of a file with CLOSE (with lock on single record)
- Unlock all records of a file with CLOSE (with lock on multiple records)
- Unlock single record with UNLOCK (with lock on single record)
- Unlock single record with UNLOCK (with lock on multiple records)
- Unlock single record by reading another one (only with lock on single record)
- Unlock single record after REWRITE (with lock on single record)
- Unlock single record after REWRITE (with lock on multiple records)

The lock management on MySQL and MS SQL Server requires a couple of peculiar configuration settings that are discussed in this documentation. See EDBI Routines for MySQL (InnoDB engine) and MariaDB and EDBI Routines for Microsoft SQL Server for details.

Note: If you're working in a Application Server (Thin Client) or File Server environment and you wish to have a full support for locking features, then you may consider handling locks through the Internal lock management.

Moving among duplicate key values

When you change the order of read (e.g. you perform a READ NEXT after a READ PREVIOUS or a READ PREVIOUS after a READ NEXT) among duplicated values of an alternate key, the EDBI routine retrieves the first record whose key value doesn't match with the current one.

A practical example follows.

Suppose to have a file with the following content:

Record	
Primary Key	**Alt. Key**
1	AAA
2	BBB
3	BBB
4	BBB

The table below lists a series of operations and tells which record is read by ISAM file handlers like JIsam or c-tree versus the record read by DatabaseBridge:

Operation	Record read by ISAM	Record read by DatabaseBridge
MOVE "BBB" TO *Alt. Key*	...	...
START KEY NOT LESS *Alt. Key*	...	...
READ NEXT	2 BBB	2 BBB
READ NEXT	3 BBB	3 BBB
READ PREVIOUS	2 BBB	1 AAA

Transactions

A transaction means a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity. For a transaction to be completed and database changes to made permanent, a transaction has to be completed in its entirety. If something happens before the transaction is successfully completed, any changes to the database must be kept track of so that they can be undone.

In order to manage transactions, the connection to the database must not be in autocommit mode, therefore the following setting should appear in the runtime configuration when the connection to the database is established:

```
iscobol.jdbc.autocommit=false
```

In this way, every modification to the database data is done within a transaction. When every modification has been done, the program can confirm (COMMIT) or cancel (ROLLBACK) and the database will be updated accordingly.

In order to confirm the data modification, your program should call the EDBI_COMMIT routine:

```
CALL "EDBI_COMMIT" USING RET-CODE, RET-ERMC.
```

In order to cancel the data modification, your program should call the EDBI_ROLLBACK routine:

```
CALL "EDBI_ROLLBACK" USING RET-CODE, RET-ERMC.
```

RET-CODE and RET-ERMC should be defined as follows:

```
01 RET-CODE   PIC S9(10).
01 RET-ERMC   PIC X(256).
```

They receive the SQL return code (0 if successful, non-zero if an error occurred) and the error description respectively.

In this scenario you can also instruct the DatabaseBridge to automatically issue a COMMIT after n operations. See -cc option for details.

Including ESQL

EDBI routines are standard COBOL programs with embedded SQL and therefore they can work together with other COBOL programs that use ESQL, sharing the connection to the database.

This possibility could help you in optimizing some procedures where the ISAM logic is too slow if compared with the SQL logic. In these cases you may consider to rewrite only those specific procedures using ESQL and leave the rest of the application working with DatabaseBridge.

Another scenario where it's useful to include COBOL/ESQL programs in the application is for the interaction with database resources that are unknown to the DatabaseBridge, for example tables that were not mapped to COBOL indexed files as well as stored procedures that you need to call for specific tasks.

The simplest scenario where DatabaseBridge can work together with COBOL/ESQL programs consists in one single connection to the database that must be shared between DatabaseBridge and COBOL/ESQL programs. A rather more complex scenario consists in an environment where multiple connections are established and therefore you have to ensure that both DatabaseBridge and COBOL/ESQL programs work on the desired connection. These two scenarios are discussed below.

Mixing DatabaseBridge and ESQL on one single connection

No particular action is required when mixing DatabaseBridge and ESQL on the same connection. The program that connects first creates the connection to the database, it doesn't matter if it's a program performing OPEN via DatabaseBridge or a COBOL/ESQL program issuing a CONNECT statement. Once the connection is established, both DatabaseBridge and COBOL/ESQL can perform queries without issues.

Mixing DatabaseBridge and ESQL in a multi-connection environment

Both COBOL/ESQL and DatabaseBridge allows you to generate multiple connections, either to the same database or to different databases.

For more information about handling multiple connections in COBOL/ESQL programs, see Working on multiple connection simultaneously in the Embedded SQL Reference manual.

For more information about handling multiple connections with DatabaseBridge programs, see Working with multiple connections in this book.

When multiple connections are involved, you have to take care of the connection names.

The default connection performed by DatabaseBridge for files that are not mapped to a particular connection and the default connection performed by COBOL/ESQL programs issuing a CONNECT statement without the AS clause is named “DEFAULT”.

When the AS clause is used in the CONNECT statement, such clause specifies the name of the connection. With reference to DatabaseBridge, instead, the name of the additional connections is extrapolated from the iscobol.easydb.connection_name.FileName configuration setting. DatabaseBridge works on the DEFAULT connection by default and then switches to the necessary separate connection when it finds a file that is mapped to that separate connection.

The good practice in this kind of scenario is to make the COBOL/ESQL program work on a dedicated named connection and switch back to the DEFAULT connection when done, so that DatabaseBridge is not influenced by the connection switch performed by the COBOL/ESQL program. E.g.

```
      *connect with a custom connection name
           exec sql
              connect to     :db  as MyCustomConnection
                      user   :usr
                      using  :pwd
           end-exec.
      *switch to this new custom connection
           exec sql set connection MyCustomConnection end-exec.
      *perform the desired SQL operations
           exec sql ...
           exec sql ...
           exec sql ...
           ...
      *switch back to the default connection
           exec sql set connection DEFAULT end-exec.
      *exit program
           goback.
```

Troubleshooting

DatabaseBridge converts SQL errors to COBOL file status where applicable. For those errors that don’t correspond to any known COBOL file status, a 9D file status is returned. The extended information returned along with the status 9D provides error description.

It’s possible to know which SQL query has been used by DatabaseBridge along with the exit status by tracing the Framework SQL activity. This kind of trace is obtained by adding 256 to the value of iscobol.tracelevel.

The following configuration, for example, traces environment settings, programs life cycle, i/o and SQL in a file named *iscobol.log* under /tmp.

```
iscobol.logfile=/tmp/iscobol.log
iscobol.tracelevel=267
```