isCOBOL Evolve: SDK User's Guide

Key Topics

- Configuration
- Compiler
- Runtime Framework
- Debugger
- RemoteCompiler
- Utilities

Copyrights

Copyright (c) 2023 Veryant
6390 Greenwich Drive, #225, San Diego, CA 92122, USA

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 prior written authorization of Veryant and its licensors, if any.

Table of Contents

1. isCOBOL Introduction 1

isCOBOL for the Java Platform 1
isCOBOL: Application Server 1
isCOBOL: User Interface 1
isCOBOL: Data Access 2
isCOBOL: Databases 2
isCOBOL: Handheld Devices 2

2. Installation 3

Installing on Windows 3
Installing on a PDA 3
Installing on UNIX or LINUX 4
Installing on Mac OSX 5

3. Configuration 8

Runtime Configuration 9

4. Compiler 11

Compiler Options 12
Compiler Properties 23
Compiler Directives 25
Source code preprocessing 28

5. Remote Compiler 29

Server configuration 29
Client configuration 32

Windows service 33

6. Runtime Framework 34

Runtime Options 34
Runtime Framework Properties 35
Keyboard Configuration 78

7. isCOBOL IDE 87

Configuration 87
Customization 87
The isCOBOL IDE Perspective 103
isCOBOL Explorer 104
Editors 105
Outline 114
Properties 115
Problems 116
Console 117
Search 117
Bookmarks 118
Tasks 118
History 119
Error Log 119
Customization 120
Working with Projects 120
Creating a new Project 120
Setting Project properties 120
Adding an existing Project to the current Workspace 123
Creating a new program 123
Adding existing programs to the Project 124
Code Editing 125
Compiling 125
Remote Compiling 126
External Preprocessors 127
Run and Debug 129
Running as Web Application 135
Import / Export Project Properties 135

Deployment facility 136
Working with Screen Programs and File Layouts 139
Creating a new Screen Program 139
Adding existing Screen Programs to the current Project 140
Screen Program structure 141
Screen Program properties 141
Repository 142
Working Storage and Linkage Section management 144
Drawing the Screen 144
Adding a new Screen to the program 149
Creating a new File Layout 150
Adding an Existing File Layout to the Current Project 151
File Layout Structure 152
File and Record Definition 153
Generating File Layouts from Existing Copybooks 157
Datasets 159
External Paragraphs and Variables 161
Code generation 161
Consistency Check 162
Screen Designer Reference 163
BAR 164
BITMAP 169
CHECK BOX 175
COMBO BOX 182
DATE ENTRY 189
ENTRY FIELD 197
FRAME 205
GRID 211
JAVA BEAN 226
LABEL 232
LIST BOX 237
PUSH BUTTON 245
RADIO BUTTON 252
SCROLL BAR 259
SLIDER 264

TAB CONTROL 270
TREE VIEW 276
WEB BROWSER 283
STATUS BAR 289
TOOL BAR 292
MENU 295
WINDOW 295
isCOBOL Tools 303
Managing isCOBOL Server 303
Managing Load Balancer 304
Managing Remote Compiler server 304
Launching isCOBOL Utilities 305
Importing programs from AcuBench 305
Importing a Program from AcuBench 306
Importing a Data Layout from AcuBench 307
Importing a Program with tagged areas from AcuBench 309
Importing programs from Totem 311
Importing a Program from Totem 312
Importing a Data Layout from Totem 314
Importing a Program with tagged areas from Totem 316

8. isCOBOL Server (Thin Client and Distributed Processing) 319

Users count 320
Connections count 320
Client and Server info 320
Usage of isCOBOL Client 321
Login 324
Tracing the Thin Client Activity 325
Tracing Application Server Activity 325
Tracing Clients Activity 326
Remote objects 326
Hook program 327
Internal lock management 329
Windows service 330
isserver.exe usage 330

Output redirection 332
Java options and Classpath 332
Server settings 332
isCOBOL File Server 332
isCOBOL File Server usage 333
Veryant ODBC 334
isCOBOL Graphical Terminal 339
isCOBOL Client Listener 339
Configuring the server 339
isCOBOL Client Listener usage 340
Configuring Putty to use isCOBOL Client Listener 340
isCOBOL Load Balancer 345
Licensing 345
isCOBOL Load Balancer usage 345
Setting up the isCOBOL Load Balancer 346
Windows Service 347

9. Veryant UDBC 350

Introduction 350
Veryant UDBC Architecture 350
Database Configuration 351
UDBC Configuration Tool 352
UDBC Server 353
VISQL 354
ODBC Driver 354
JDBC Driver 355
Troubleshooting 355

10. Utilities 357

COBFILEIO 357
Configuration Properties 358
API Reference in Javadoc Format 358
GIFE 359
ISCONFIG 361
ISMIGRATE 361
JDBC2FD 365

JUTIL 367
JIsam file generation 368
Micro Focus file conversion 369
XML2WRK 370

11.Visual Debugger 372

Remote Debugging 372
The Debugger Window 373
Menu Bar 374
Toolbar 375
Source Area 375
Output Window 375
Command Area 375
Information Window 375
Debugger Functions 376
Pop-up menu 390
Debugger Properties 391

12.Wrappers 392

Standard wrappers 392
Windows wrappers 393
The -J option 394

13.Special Features 396

New syntax 396
I/O 397
Routines and functions 398
Distributed environment (Application Server) 398
GUI 399
Debugger 403

14.Index 405

Chapter 1

Compiler and Runtime

Overview

The isCOBOL Compiler is a platform agnostic, ANSI-compliant COBOL compiler that generates object-orientated code which efficiently runs on any platform that supports a Java Runtime Environment (JRE) version 8, 11 or 17. Because the isCOBOL Compiler is written 100% in Java, one COBOL compiler can be used to develop, deploy and test on a wealth of platforms including AIX, HP-UX, Linux, Solaris, Windows and mainframe systems.

Written 100% in Java, the isCOBOL Runtime Environment enables applications to able to run on any device supporting a Java Virtual Machine (JVM) -- from mainframes to mobile phones -- and this includes application logic, user interface, and data access.

Getting Started

The setup of a Compiler and Runtime 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 Compiler and Runtime. For best results and performance, install the latest JDK version available for your platform. isCOBOL is certified to work correctly with both Oracle JDK and OpenJDK from version 8 to version 17.

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

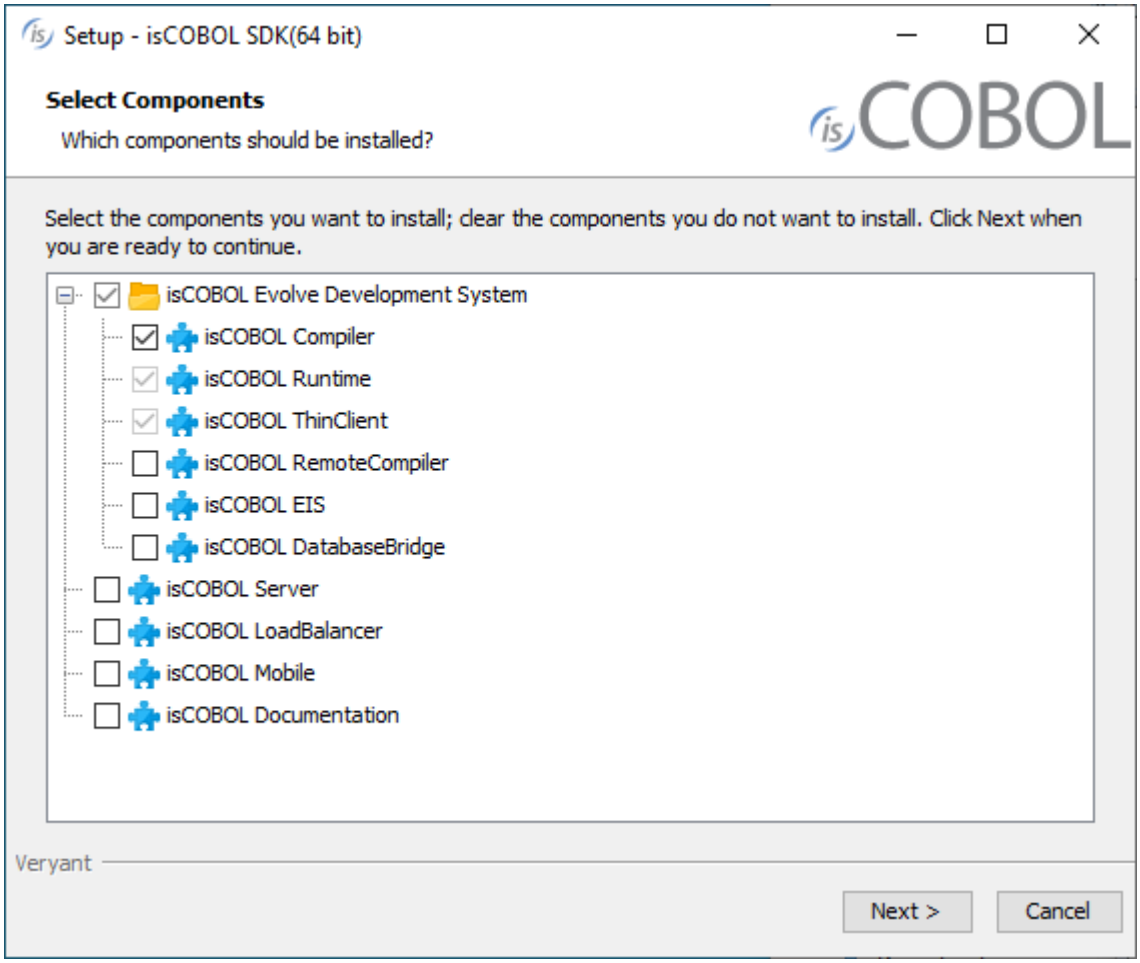
Windows

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .

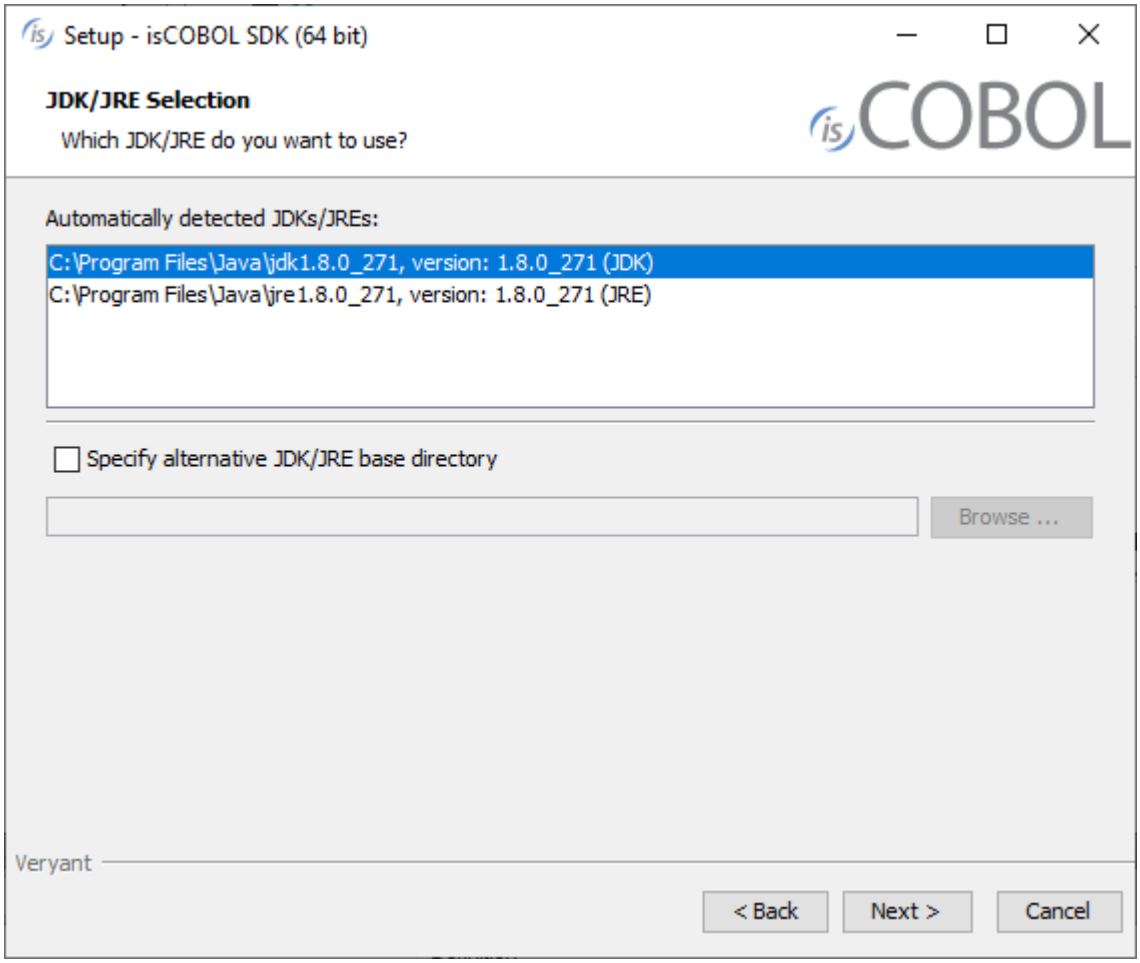
2. Go to "https://support.veryant.com".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the list of files for Windows x64 64-bit or Windows x86 32-bit. Select isCOBOL_2023_R1_*n*_Windows.*arc*.msi, where *n* is the build number and *arc* is the system architecture.
6. Run the downloaded installer to install the files.

Note - If your Windows has the option "Run as Administrator", you should run the setup with that option, otherwise the setting of environment variables might silently fail. Environment variables setting is not necessary if you work from the isCOBOL Shell (explained later).

7. Select "isCOBOL Compiler and Runtime Environment" from the list of products when prompted.



8. Select your JDK when prompted



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.

Quiet mode

The isCOBOL SDK setup supports the msi quiet mode. Settings can be driven with a response file.

A response file is a text file with name-value pairs that represent installer variables.

A response file is generated automatically after an installation is finished. The generated response file is found in the *.install4j* directory of the isCOBOL SDK and is named *response.varfile*.

When an installer is executed, it checks whether a file with the same name and the *.varfile* extension can be found in the same directory and loads that file as the response file. For example, if an installer is named *foo_setup.msi* on Windows, the response file next to it has to be named *foo_setup.varfile*.

For more information about msi setups and their command line options, see Microsoft Standard Installer Command-Line Options.

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://support.veryant.com".
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_2023_R1_*_Linux.32.i586.tar.gz
tar -xvf isCOBOL_2023_R1_*_Linux.32.i586.tar
```

   on Linux 64 bit:

```
gunzip isCOBOL_2023_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2023_R1_*_Linux.64.x86_64.tar
```

   on FreeBSD:

```
gunzip isCOBOL_2023_R1_*_FreeBSD.64.tar.gz
tar -xvf isCOBOL_2023_R1_*_FreeBSD.64.tar
```

   on Mac OSX:

```
gunzip isCOBOL_2023_R1_*_MacOSX.64.x86_64.tar.gz
tar -xvf isCOBOL_2023_R1_*_MacOSX.64.x86_64.tar
```

   on SunOS:

```
gunzip isCOBOL_2023_R1_*_SunOS.64.tar.gz
tar -xvf isCOBOL_2023_R1_*_SunOS.64.tar
```

7. Change to the "isCOBOL2023R1" folder and run "./setup", you will obtain the following output:

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

                           isCOBOL EVOLVE Installation
                           For isCOBOL Release 2023R1
                       Copyright (c) 2005 - 2023 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)........ (no)
    [8] isCOBOL LoadBalancer............................. (no)
    [9] isCOBOL Mobile................................... (no)

Install Path:
    [P] isCOBOL parent directory: UserHome/veryant

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 ]
```

The following text depends on the current environment:

UserHome	current user home directory
JavaHome	current JDK/JRE directory detected by the setup script

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

The setup script might not be available for your Unix platform or you might want to avoid it. In this case you can just extract the tgz in the destination folder. If you do in this way, then the following environment variables must be set in the system in order to compile, run and debug: ISCOBOL_JDK_ROOT (or ISCOBOL_JRE_ROOT), ISCOBOL, LD_LIBRARY_PATH and PATH.

ISCOBOL_JDK_ROOT	root directory of a Java JDK. I'ts required to compile, run and debug
ISCOBOL_JRE_ROOT	root directory of a Java JRE. Can be used instead of JDK if you don't need to compile
ISCOBOL	root directory of isCOBOL. The directory where you extracted the tgz

LD_LIBRARY_PATH	the isCOBOL native/lib directory must be added here
PATH	The isCOBOL bin directory must be added here

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 LD_LIBRARY_PATH=$ISCOBOL/native/lib
export PATH=$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://support.veryant.com".
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_2023_R1_*n*_noarch.tar.gz, where *n* is the build number.
6. Extract all contents of the archive:

```
gunzip isCOBOL_2023_R1_*_noarch.tar.gz
tar -xvf isCOBOL_2023_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.

The isCOBOL Compiler looks for the following configuration properties for license keys:

```
iscobol.compiler.license.2023=<license_key>
iscobol.license.2023=<license_key>
```

The isCOBOL Runtime looks for the following configuration property for license keys:

```
iscobol.license.2023=<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. a custom configuration file passed on the command line
5. %ISCOBOL%\iscobol.properties

Unix/Linux

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

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.

Compiler

Overview

The job of any compiler is to convert human-readable source code to an object that a computer can run. To accomplish this, isCOBOL performs the following steps:

1. The COBOL source code is translated to a Java source code.
   The name of the Java source file is obtained by converting the name of the COBOL source file to upper-case and replacing dashes by underscores. For example, A COBOL source file whose name is *Hello-World.cbl* generates *HELLO_WORLD.java*.
2. If no errors occur, then Compiler looks for an existing class file (e.g. *HELLO_WORLD.class*) and deletes it, if it exists.
3. Finally, the Compiler compiles the Java source code generating a Java class file. The Java source code is then deleted unless you use the -jj option.

There are situations in which more than one class file is generated.

- If the source code contains PERFORM THREAD statements, a class file for each thread is generated in addition to the program class file. These classes are named using a progressive number (*program*$1, *program*$2, ..., etc).
- If the source code contains the THREAD-LOCAL-STORAGE SECTION or the IS THREAD-LOCAL clause, an additional class is generated. This class is named using the "_my_thloc_stg" suffix (i.e. *program$_my_thloc_stg*).
- If the program is object-oriented, a class file for each method is generated in addition to the program class. These classes are named using the method name (i.e. *program$method*).
- If the -big compiler option is used, some class files are generated in addition to the program class file. These classes are named using the program name, the “inner” and “CONST” keywords, and progressive numbers. The –big behavior can be configured by the iscobol.compiler.max_constants * and iscobol.compiler.max_paragraphs * properties.
- If the program contains ESQL statements with more than 700 host variables, additonal classes are generated (each containing a maximum of 700 variables). These classes are named using a progressive number (program$1, program$2, ..., etc). This behavior can be configured by the iscobol.compiler.max_hostvars * property.
- If the source code contains SORT-RETURN and/or SORT-MESSAGE special registers, an additional class is generated. The class is named *program*$SortAbort.

The command to execute the compiler is:

```
iscc Options SourceCode
```

Note: On Windows this command should be launched from inside the isCOBOL Shell. Otherwise you need to set the ISCOBOL and ISCOBOL_JDK_ROOT environment variables before using iscc.

Compiling multiple source files at once

The isCOBOL compiler supports the * wildcard in the *SourceCode* parameter.

For example, the following compilations:

```
iscc prog1.cbl
iscc prog2.cbl
iscc prog3.cbl
iscc prog4.cbl
```

can be done all at once with the command:

```
iscc prog*.cbl
```

When you compile multiple source files with one command and a COBOL error occurs (for example "Unexpected token"), the Compiler continues to the next source file. However, if a Java error occurs (for example "code too large"), the Compiler stops and won't compile subsequent source files.

Automatic compilation of referenced COBOL classes

If a COBOL program references a COBOL class with object oriented syntax, the COBOL class is automatically compiled if necessary.

Consider the following source files, for example:

prog1.cbl

```
       program-id. prog1.
       configuration section.
       repository.
           class class1 as "class1".
       procedure division.
       main.
           invoke class1 "method1".
           goback.
```

class1.cbl

```
       class-id. class1 as "class1".
       identification division.
       factory.
       procedure division.
       identification division.
       method-id. method1 as "method1".
       procedure division.
       main.
      *    method code here
       end method method1.
       end factory.
```

When *prog1.cbl* is compiled, if *CLASS1.class* is not found in the Classpath, then the Compiler tries to compile *class1.cbl* before proceeding with the compilation of *prog1.cbl*.

This feature can be disabled by adding the -noarcc option to the Compiler command line.

Exit status

The Compiler returns 0 if the compilation is OK and a number greater than 0 if the compilation fails. When you compile multiple source files at once, if any of these files produce severe errors, then the return code will be greater than zero.

The following table lists the possible exit status codes returned by the Compiler:

Exit Code	Meaning
0	No errors
1	A Java exception occurred during the compilation process (e.g. ClassNotFoundException: com.sun.tools.javac.Main)
4	Compilation failed due to Severe errors
5	Invalid command line (e.g. unsupported option)

Compiler Options

The -help option displays all available options:

Common Options

`-b`	Treat characters as bytes in STRING, UNSTRING, and INSPECT statements. By default isCOBOL internally converts strings into Unicode. The -b option makes isCOBOL work directly on the string without any conversion. This approach increases performance but programs compiled with this option may not work correctly if they use national items. The option takes effect on standard alphanumeric data items, while national data items are not affected.
`-c=config_file`	Use the configuration file identified by *config_file*. See Configuration for the list of the configuration properties that are applicable to the Compiler and for details about how the configuration is built.
`-conly=config_file`	Use only the configuration file identified by *config_file*. See Configuration for the list of the configuration properties that are applicable to the Compiler.
`-d`	Include debug information.
`-dx`	Enable extended debugger functions. This option implies -d. In addition to the standard debug features, all the variables in the class are generated, including those not used in the program and the literal constants that are generated during the execution and not as static fields in the generated class. When a program is compiled with -dx, the Debugger is able to query and set all the items of the program Data Division including the items that are not used in the Procedure Division and the IDE allows the source code to be changed while debugging. With -dx the Debugger is able to skip statements through the "jump" command.
`-edc`	Removes output class files if compilation fails. This option works only when the source file has been recognized as a COBOL program or class. If some of the class files cannot be deleted for some reason, the compiler doesn't signal it. The number of removed classes is shown at the bottom of the compiling result.
`-ef`	Output errors to a '.err' file. The file has the same name as the source and is created only if there are compiler errors or warnings. The Compiler automatically removes the '.err' file before starting to compile the source, so existing err files disappear after a correct compilation. The Compiler output is also traced on the system output.
`-eo=DirName`	Specifies the directory for error files. If the directory does not exist or doesn't have the correct permissions, the err file will not be created.
`-es`	Stop compilation and return a non-zero exit code if an error occurs. This option is useful when compiling multiple sources at once.
`-esme=n`	Sets the maximum number of errors printable by the Compiler to *n*, where *n* is a positive number. When multiple source files are compiled at once (e.g. if you use wildcards in the source name), the option limits the number of errors for each single source file, not for the whole compilation.
`-help`	Display the list of all compiler options with a quick explanation for each one of them and exit.
`-helpx`	Display the list of all compiler options including experimental options with a quick explanation for each one of them and exit.

`-noarcc`	Disable the Automatic compilation of referenced COBOL classes
`-oe`	Optimize EVALUATE with string literals. When a EVALUATE statement tests string literals, isCOBOL uses the Java SWITCH statement instead of the EVALUATE implementation.
`-v`	Display the Compiler version number and exit.
`-verbose`	Display verbose output, e.g. the count of errors, informational and warnings.
`-sysc`	Allows you to override COBOL library routines. This option is useful for overriding COBOL library routines such as C$SYSTEM and for speeding up CALLs to subprograms that are called frequently. The -sysc option causes the compiler to place the Java class in the com.iscobol.lib package (or com.iscobol.lib_n package when compiling with -cp). The isCOBOL runtime framework searches for programs in this package before searching other places such as paths specified in iscobol.code_prefix. To execute programs compiled with -sysc, they must be found in a path or jar file listed in the Java class path. With the -sysc option, the compiler adds "package com.iscobol.lib;" (or "package com.iscobol.lib_n;") to the top of the generated Java source code for the program class. For example, if a program named MYPROG is compiled with -sysc then the generated Java class will be named com.iscobol.lib.MYPROG. The -sysc option should be used only for CALLed subprograms. If it is used for main programs or COBOL objects there is no error at compile time, but the generated object cannot be executed.

Compatibility Options

`-ca`	Acucobol compatibility flag: • Use of the ALLOWING clause in OPEN statements is supported. • Different INSPECT TALLYING behavior (see Language Reference for details). • STRING dest-item can be JUSTIFIED • UNSTRING delimiter can be a numeric USAGE DISPLAY item • NEXT SENTENCE statement • Different handling of END-ACCEPT
`-caec`	WITH CONVERSION is assumed for MOVEs from alphanumeric items to edited items.
`-cax`	Specifies the default file assignment as external.
`-ccbas`	Count bytes instead of characters in FIXED/ANSI source files. This option is useful during the compilation of source files that include double-byte characters. Without this option, text written after column 72 may be considered as written before column 72 due to the double-byte characters in the source line.
`-ccmf`	Pass literals to the called program with Micro Focus rules. BY REFERENCE: A numeric literal is passed as an integer 32-bit comp-x item if it is not negative or as an integer 32-bit comp (binary) item if it is negative, even if the literal is not an integer. A decimal point in the numeric literal is ignored. An alphanumeric literal is passed in the same way as when the BY CONTENT clause is used. BY VALUE: If a numeric literal is specified, then the data description of the item is equivalent to a signed numeric item USAGE BINARY. BY CONTENT: if a literal is specified, then the implied data description of the item is equivalent to an alphanumeric data item with the same size as the literal and with its contents set to the value of the literal. **Note -** this option affects also the calls to library routines.
`-cdlz`	Shows USAGE DISPLAY memory content. This option affects the internal definition of variables. If external variables are used, then all programs have to be compiled with this option, otherwise a mismatch error is received. This option shouldn't be used with programs that display a graphical user interface. Note that this option affects also the debugging of the program, as the Debugger will show memory content of USAGE DISPLAY items instead of showing their value.
`-cfl`	Compatibility setting for file SEQUENTIAL is LINE SEQUENTIAL. When this is not set, files with ORGANIZATION IS SEQUENTIAL are treated as BINARY SEQUENTIAL.

`-cfp36`	Intermediate results are always calculated to 36 digits. isCOBOL arithmetic uses 3 types of number: (A) fixed point numbers with number of digits less or equal to 18; (B) fixed point numbers with number of digits greater than 18; (C) floating point numbers; Without this option, when an arithmetic operation occurs, the most wide type is used to perform the operation, i.e: (A) +-*/ (A) -> (A) (A) +-*/ (B) -> (B) (B) +-*/ (B) -> (B) (A) +-*/ (C) -> (C) (B) +-*/ (C) -> (C) (C) +-*/ (C) -> (C) The option causes that the two operation whose result can be wider than the operands, i.e. division and multiplication, will be performed using the (B) type, i.e.: (A) */ (A) -> (B) (A) */ (B) -> (B) (B) */ (B) -> (B) (A) */ (C) -> (C) (B) */ (C) -> (C) (C) */ (C) -> (C)
`-cko`	List keys in offset order. Without this option keys are listed following the order they're declared in the FILE-CONTROL paragraph. It changes the order in which keys are registered in the physical file, that can be verified with file management utilities such as jutil and ctutil. As a consequence, this option affects XML/ISS dictionaries as well as the I$IO and file interfaces where keys are pointed by ordinal number.
`-ci`	ICOBOL compatibility. • LOCK MODE IS MANUAL WITH LOCK ON MULTIPLE RECORDS is implied unless -cm option is used as well. • alternate keys are considered WITH DUPLICATES by default.
`-clk0`	Linkage items are unlinked when the program is called.

-cm	Micro Focus compatibility flag. It supports the following: • NEXT SENTENCE statement • if LOCK MODE is omitted, opening a file causes it to become EXCLUSIVE, unless the file is opened for INPUT. • duplicated constants definitions • the syntax H"xx" is treated as a number instead of a string • items that are not Usage Display can be used in UNSTRING statements • EQUALS, IS UNEQUAL TO and EXCEEDS operators • STRING dest-item can be JUSTIFIED • UNSTRING dest-item is not required to be USAGE DISPLAY • UNSTRING source-item can be numeric-edited • LENGTH OF is internally represented as COMP-5 • b"binaryValue" and o"octalValue" syntax • REWRITE allowed on print files • BEFORE 1 assumed for WRITE without ADVANCING clause on line sequential files • COMP-1 is translated to FLOAT. • COMP-2 is translated to DOUBLE. • OCCURS indexes are not considered EXTERNAL even if they're declared within an EXTERNAL group item. • ability to omit the index when referencing OCCURS data items (the first occurrence is assumed) • different handling of literals passed in a CALL statement In addition, SORT RETURN and SORT MESSAGE internal variables are created. SORT-MESSAGE is never used while SORT-RETURN is checked before every RELEASE and RETURN statement and if it contains 16 the sort is aborted and the control returns to the instruction following the SORT statement.
-cms	Microsoft COBOL compatibility flag. It supports the following: • ACCEPT (line, column) identifier • DISPLAY (line, column) identifier | literal | ERASE • the usage COMPUTATIONAL (COMP) is equivalent to usage DISPLAY • new usage COMPUTATIONAL-0 (COMP-0) that is equivalent to SHORT In addition, LIN and COL internal variables are created. As a consequence the reserved word COL cannot be used.
-cndbcs	Use DBCS instead of Unicode in PIC N without USAGE NATIONAL
-cnlz	Leading zeros are shown when numeric data items are displayed on a character based screen.
-cod0	Changes OCCURS DEPENDING ON behaviour. This affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING clause, but not subordinate to it. With -cod0, these items have fixed addresses, and begin after the end of the space allocated for the table at its maximum length. This is the default behavior if the -cod1 option is not used.
-cod1	Changes OCCURS DEPENDING ON behaviour. This affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING clause, but not subordinate to it. With -cod1, these items always immediately follow the table, regardless of its current size; this means their addresses change as the table's size changes.

-coe	Closes all files opened by the program when the program exits. This option has priority over the iscobol.file.close_on_exit (boolean) configuration setting. For example, if a program is compiled with -coe, files will be closed even if it sets iscobol.file.close_on_exit=false with a SET ENVIRONMENT statement before exiting.
-cp	Enable full pointer support. Use this option if you need to call C functions that reuse pointers. Programs compiled with this option can run along with programs compiled without this option in the same COBOL application, but they cannot share parameters each other. Configuration properties can be used to share information between these two different kind of programs. Programs compiled with this option can't call C functions with CALL CLIENT statements in a thin client environment. If programs try to allocate or reference items out of their bounds, consider using -m1 as well, otherwise the whole JVM may crash. Pointers are 4 bytes in size by default. Use -d64 to make pointers 8 bytes in size.
-cpanv	Allow ++INCLUDE statements. These statements are internally translated to COPY statements.
-cr	RM/COBOL compatibility flag. It supports the following: • DISPLAY WINDOW-CONTROL-BLOCK syntax • DISPLAY statement without LINE clause • ERASE without EOS or EOL clauses moves the cursor at line 1 position 1 • PROGRAM-ID special register • NEXT SENTENCE statement • LINE + 1 assumed for DISPLAY where only POS is specified
-crlk	RM style lock mode: any READ LOCK on a file that does not have an applicable DECLARATIVE section is automatically translated into READ LOCK WAIT.
-crko	List keys in offset order in RM/COBOL compatibility. Without this option keys are listed following the order they're declared in the FILE-CONTROL paragraph. This option differs from -cko on split keys composed of multiple segments. When using -cko, only the first segment of the key is evaluated for the offset. When using -crko, instead, all the segments of the key are evaluated for the offset. This option changes the order in which keys are registered in the physical file, that can be verified with file management utilities such as jutil and ctutil. As a consequence, this option affects XML/ISS dictionaries as well as the I$IO and file interfaces where keys are pointed by ordinal number.
-crv	Compatibility setting for implicit record varying size for files with multiple record definitions with different lengths and files whose record is varying from size to size. When this is not set, files are treated as fixed length and the maximum record length is used. This option affects files that include two or more record definitions with different size as well as files that include OCCURS DEPENDING.

`-csdb2`	Treat ESQL in compatibility with IBM DB2.
`-csl`	Treat the COBOL name in ASSIGN clause as a literal. This allows a mapping to be created for the file name if the iscobol.file.env_naming (boolean) configuration property is set to true.
`-csora`	Treat ESQL in compatibility with Oracle Pro*COBOL.
`-csqn`	Compatibility setting: SQL returns an error if a host variable is set to null. The error number stored in SQLCODE is 1405. It can be customized by setting iscobol.esql.value_sqlcode_on_null, but not by setting *iscobol.esql.sqlcode.1405*.
`-csqq`	Quotes inside ESQL statements are left as they are by the Compiler. Without this option, all kinds of quotes are translated to single quotes by the Compiler.

`-cudc`	Treats numeric USAGE DISPLAY data as characters in comparisons and moves.
	
	This option affects the comparison between numbers whose usage is DISPLAY in particular cases, using a byte by byte comparison instead of comparing the numeric representation.
	
	The byte by byte comparison is used when:
	
	• two unsigned numbers with usage DISPLAY with the same length and the same number of decimal digits are compared
	• an unsigned integer number and an alphanumeric elementary item with the same length are compared
	• an unsigned number with usage DISPLAY is compared with ZERO (ZEROES ZEROS) or 0. In this case the comparison is made comparing each digit within the number, byte by byte, with the character '0'.
	• an unsigned number with usage DISPLAY is compared with SPACE (SPACES) or " ". In this case the comparison is made comparing each digit within the number, byte by byte, with the character " ".
	
	This option affects the MOVE statement when one operand is an unsigned numeric data item USAGE DISPLAY, a numeric constant or a numeric literal and the other one is an unedited alphanumeric item.
	
	• When the sender operand is an alphanumeric data item and the receiver operand is a numeric data item USAGE DISPLAY, then a byte by byte move is performed as if the sender operand should contain only one digit, that is the string representation of an integer number, but no check is performed on the real content: e.g.:
	
	`MOVE "FA" TO PIC-XX`
	`MOVE PIC-XX TO PIC-9 = A`
	`MOVE PIC-XX TO PIC-99 = FA`
	`MOVE PIC-XX TO PIC-999 = 0FA`
	`MOVE PIC-XX TO PIC-Z = A`
	`MOVE PIC-XX TO PIC-ZZ = FA`
	`MOVE PIC-XX TO PIC-ZZZ = 0FA`
	`MOVE PIC-XX TO PIC-V9 = 0`
	`MOVE PIC-XX TO PIC-9V9 = A0`
	`MOVE PIC-XX TO PIC-9V99 = A00`
	
	• When the sender operand is an unsigned numeric USAGE DISPLAY data item and the receiving operand is a non edited alphanumeric data item, then a byte by byte move is performed as if the first operand were an alphanumeric item itself.
	• When the sender and the receiver operands have an identical PICTURE and USAGE, a byte by byte move is performed as if both operands were alphanumerics

`-cv`	IBM COBOL compatibility flag. It supports following syntaxes: • EXAMINE • EXHIBIT • EJECT • SKIP • IF OTHERWISE • NOTE • TRANSFORM • ADVANCING (WRITE statement) • AFTER POSITIONING • USE GIVING • TIME-OF-DAY • WHEN-COMPILED • WRITE ADVANCING *Special-Name* • VALUE OF • CURRENT-DATE • RECORDING MODE (FD clause) • AFP-5A, C01 and CSP in Special-Names • *numeric* FILE STATUS • *multiple* FILE STATUS • PROCESS and CBL directives • MOVE with multiple TO keywords • USE FOR DEBUGGING and WITH DEBUGGING MODE • Occurs indexes are initialized to 1. • Occurs indexes are not considered EXTERNAL even if they're declared within an EXTERNAL group item. • Ability to omit the index when referencing OCCURS data items (the first occurrence is assumed). • Characters before the last hyphen in the name of files assigned to EXTERNAL are ignored. • SORT RETURN and SORT MESSAGE internal variables are created. SORT-MESSAGE is never used while SORT-RETURN is checked before every RELEASE and RETURN statement and if it contains 16 the sort is aborted and the control returns to the instruction following the SORT statement. • COMP-1 is translated to FLOAT. • COMP-2 is translated to DOUBLE. • SYNCHRONIZED clause also affects group items.
`-cva`	IBM arithmetic compatibility. If the dest-item of a calculation does not include the decimal part, the result of internal operations made to set that result lose their decimal part as well. The option affects both numeric and numeric-edited data items. For example: `compute res = (11/4) * 4` If res is declared as PIC 99 or PIC Z9, it will be set to 8 If res is declared as PIC 99v99 or PIC Z9.99, it will be set to 11.00 If the -cva option is not used, the result of the above calculation will always be 11.

`-cva2`	IBM arithmetic compatibility with powers treated differently. If the expression of the COMPUTE statement contains a power whose exponent is: • a number • a data-item with decimals • an expression containing only numbers or data-items with decimals then standard -cva rules are not applied. For all other cases, the behavior is the same as when the -cva option is set.

Data Options

`-align=number`	Allows you to specify the data alignment modulus. For example, "-align=8" specifies that data should be aligned on eight-byte boundaries. The default value is 1
`-d1`	Binary data whose length is <= 2 are stored in 1 byte
`-d5`	Treat BINARY as COMP-5. Note that only items explicitly defined as BINARY are affected. COMP and COMP-4 are not affected despite they're equivalent to BINARY. In order to treat COMP and COMP-4 as COMP-5, use -rm=newmeaning,word..., e.g. -rm=COMP-5,COMP-4.
`-d64`	Use 64-bit pointers for USAGE POINTER data items. This option should be used only in conjunction with -cp. Without this option, pointers are 4 bytes in size. With this option, pointers are 8 bytes in size.
`-dca`	Use ACUCOBOL numeric format. The compiler uses this convention by default if no other convention is specified. See USAGE clause for details about how numeric data items are affected by this option.
`-dcb`	Use MBP COBOL numeric format See USAGE clause for details about how numeric data items are affected by this option.
`-dcd`	Use Data General numeric format for binary items. See USAGE clause for details about how numeric data items are affected by this option.
`-dcdm`	Store any data item whose underlying type is binary in the minimum number of bytes needed to hold it. Normally, binary types are stored in two, four, or eight bytes. This option can be used to emulate the ACUCOBOL -dm option.
`-dci`	Use IBM sign encoding and IBM COMP sizes. COMP sizes are 1 (only if -d1 option is also used), 2, 4, 8, 12 or 16 depending on the item picture. See USAGE clause for details about how numeric data items are affected by this option.
`-dcii`	Use IBM sign encoding and IBM COMP sizes. COMP sizes are 2, 4, 8 or 16 depending on the item picture. See USAGE clause for details about how numeric data items are affected by this option.
`-dcm`	Use Micro Focus sign encoding and Micro Focus COMP sizes. See USAGE clause for details about how numeric data items are affected by this option.

`-dcmi`	Use Micro Focus sign encoding and IBM COMP sizes (like MF -C IBMCOMP). This option has the same effect of -dcm except that the length of COMP items is calculated in the same way as -dca. See USAGE clause for details about how numeric data items are affected by this option. In addition, the SYNCHRONIZED clause affects also group items.
`-dcn`	Use NCR COBOL numeric format. See USAGE clause for details about how numeric data items are affected by this option.
`-dcr`	Use Realia sign storage convention. Sign information for S9(n) variables is stored using the conventions for Realia COBOL, and their conversion to binary decimal is the same as that performed by the Realia compiler.
`-dcv`	Use numeric sign formats that are compatible with VAX COBOL. These are identical to the IBM formats, except that unsigned COMP-3 fields place X"0C" in the sign position, instead of X"0F".
`-di`	Initialize values of WORKING-STORAGE SECTION data items and indexes by type. • numeric items are initialized to zero (overriding existing initialization set through the -dv option) • numeric-edited data items are initialized as follows: digits in the position of 9 symbols are initialized to zero, other digits are initialized to space; currency symbols and separators are preserved • alphabetic and alphanumeric items are initialized to the value specified using the -dv option or ASCII spaces if -dv is omitted • occurs indexes are initialized to 1 This option does not affect items declared with VALUE or EXTERNAL clauses or those subordinate to a REDEFINES phrase.
`-dia`	Initialize values of WORKING-STORAGE SECTION data items and indexes by type. • numeric items are initialized to zero (overriding existing initialization set through the -dv option) • numeric-edited data items are initialized to spaces • alphabetic and alphanumeric items are initialized to the value specified using the -dv option or ASCII spaces if -dv is omitted • occurs indexes are initialized to 1 This option does not affect items declared with VALUE or EXTERNAL clauses or those subordinate to a REDEFINES phrase.
`-ds`	USAGE DISPLAY numeric items with no SIGN clause are treated as if they were described with the SIGN IS TRAILING SEPARATE clause.

-dv=*char*	Initialize each otherwise undefined byte in WORKING-STORAGE SECTION and FILE SECTION to the specified value when a program is first loaded or canceled and then called. *char* is the decimal representation of the character. For example, to fill the item memory area with 'A' use -dv=65. Use -dv=0 for low-values and -dv=32 for ASCII spaces. Note that the -dv option does not affect data items declared with VALUE or EXTERNAL clauses. In order to initialize EXTERNAL data items, use either -dvext or -dvexta. If -dv is omitted then the compiler behaves as if -dv=32 was specified (i.e. data items specified without a VALUE clause are filled with ASCII spaces by default). Note that when compiling with -di the value specified with -dv affects only alphabetic, alphanumeric, alphanumeric edited and numeric edited items including those that are declared FILLER. When used with -di, -dv does not affect numeric, pointer or index items. See the -di option for more information.
-dvext=*char*	Initialize each otherwise undefined byte of EXTERNAL data items in WORKING-STORAGE SECTION and FILE SECTION to the specified value when a program is first loaded or canceled and then called. *char* is the decimal representation of the character. For example, to fill the item memory area with 'A' use -dvext=65. Use -dvext=0 for low-values and -dvext=32 for ASCII spaces. If -dvext and -dvexta are omitted then the compiler behaves as if -dvext=0 was specified (i.e. EXTERNAL data items specified without a VALUE clause are filled with low-values by default).
-dvexta=*char*	Initialize each otherwise undefined byte of EXTERNAL data items in WORKING-STORAGE SECTION to the specified value when a program is first loaded or canceled and then called. *char* is the decimal representation of the character. For example, to fill the item memory area with 'A' use -dvexta=65. Use -dvexta=0 for low-values and -dvexta=32 for ASCII spaces. If -dvext and -dvexta are omitted then the compiler behaves as if -dvext=0 was specified (i.e. EXTERNAL data items specified without a VALUE clause are filled with low-values by default).
-dz	Relax size-checking rules. When this option is in effect, the values that can be held in binary and packed-decimal data types are limited only by the number of bytes of storage. The picture is not used for determining the largest value that these types can hold, and when moving to a nonnumeric destination the largest possible value determines the number of digits moved.
-dznt	Relax size-checking rules in compatibility with Micro Focus NOTRUNC directive. When this option is in effect, the values that can be held in binary data types are limited only by the number of bytes of storage. However, the PICTURE is used when moving data from a binary number to a nonnumeric data item.

`-dzta`	Relax size-checking rules in compatibility with Micro Focus TRUNC"ANSI" directive. Each numeric data item stores values up to its PICTURE in size, but COMP-5 items ignore the PICTURE when determining the largest value they can hold. However, COMP-5 items do use their PICTURE when moving a value to a nonnumeric data item.

External File Options

`-efa`	Create the External File Description XML file(s) for all the files described in the program.
`-efc`	Create the External File Description ISS file(s) for the indexed files described in the program. ISS files are required by some ctutil functions when working with c-tree. The iscobol.sqlserver.iss (boolean) feature needs these files as well. See External File Description dictionaries for more information.
`-efd`	Create the External File Description XML file(s) for the indexed files described in the program. See External File Description dictionaries for more information.
`-efo=`*`DirName`*	Specifies the directory for EFD files. If the directory does not exist or doesn't have the correct permissions, the EFD file will not be generated.

File Options

`-fl`	Single record locking is default for files WITH ROLLBACK. Normally, WITH ROLLBACK causes multiple locking rules to be in effect for a file. When this option is used, the WITH ROLLBACK clause does not affect whether single or multiple record locking rules are followed. Single locking becomes the default. You may enable multiple locking either by specifying WITH LOCK ON MULTIPLE RECORDS in a file's SELECT statement or by using APPLY LOCK-HOLDING ON file in the I-O CONTROL paragraph.
`-flsu`	Specifies a Unicode-enabled sequential access mode file handler for LINE SEQUENTIAL files. With this option, text files are read and written using the Java classes java.io.FileReader and java.io.FileWriter which access files sequentially rather than in random access mode and also preserve Unicode characters. This option should be used when reading or writing device files and pipes (i.e. files that are not disk files). This is to avoid illegal operations and to properly convert between the Java internal format (i.e. Unicode) and the desired external format. Note that the external format can be specified using the Java file.encoding property. The -flsu option is also useful when programs share sequential files between platforms with different line separators (e.g. program A creates the file on Linux and program B must be able to read the file on Windows). The -flsu option causes sequential files to be assigned to PRINT if no other assignment is specified. When using this option, OPEN I-O, REWRITE and READ PREVIOUS are not supported for LINE SEQUENTIAL files.
`-fm`	LOCK MODE IS MANUAL is implied. This option has priority over -cm and -ci in terms of default lock mode.

`-fsv`	All RECORD SEQUENTIAL files have variable-length records. The Compiler assumes that the FD includes RECORD CONTAINS 1 TO *n* CHARACTERS clause, where *n* is the length of the largest record description in the FD. Explicit RECORD or VARYING clauses in the FD are ignored.

Java Options

`-jc`	Generate the '.class' file. This is the default behavior unless -jj option is used.
`-jj`	Generate the '.java' file. By default the .java file is removed after a correct compilation. Use this option to keep the .java file on disc. If this option is not used in conjunction with -jc, the Compiler will generate only the .java file and not the .class file.
`-jo=`*`Option...`*	Passes the specified options to the 'javac' compiler. Multiple values must be separated by spaces. Example: `-jo="-source 1.8 -target 1.8"` See also iscobol . compiler . javac . options

Listing Options

`-la`	Output full listing to a '.list' file in ANSI fixed format. The list file contains all the source code, all the copybooks are merged into it (unless the SUPPRESS clause is used in the COPY statement) and in most of the cases it can be compiled as it is a standard COBOL program. If used along with -ld, only the source part is generated in ANSI fixed format, the datamap is always in FREE format. This option is guaranteed to work correctly only if the original program is already in ANSI fixed format, otherwise results are unpredictable. **Note:** The listing is generated before the syntax analysis and every dot out of quotes is considered as end of the statement, therefore, if you don't enclose copybook names between quotes, you might obtain an uncompilable list file.
`-ld`	Output full listing and data map to a '.list' file. The list file contains all the source code, all the copybooks are merged into it (unless the SUPPRESS clause is used in the COPY statement). The datamap information is stored at the bottom of the list file and provides the following information for each data item described in the program Data Division: source line, item name, offset (in case the item is part of a group item), physical length, section in which the item is defined, type flags, item type and how the item is referenced in the Procedure Division. **Note:** The listing is generated before the syntax analysis and every dot out of quotes is considered as end of the statement, therefore, if you don't enclose copybook names between quotes, you might obtain an uncompilable list file.
`-lf`	Output full listing to a '.list' file. The list file contains all the source code, all the copybooks are merged into it (unless the SUPPRESS clause is used in the COPY statement) and in most of the cases it can be compiled as it is a standard COBOL program written in free format. **Note:** The listing is generated before the syntax analysis and every dot out of quotes is considered as end of the statement, therefore, if you don't enclose copybook names between quotes, you might obtain an uncompilable list file.

`-lfo`	Creates only a full listing of the program. This option is the same as -lf except that the Compiler doesn't compile to a Java class, it just generates the listing file and exits.
`-lo=`*`DirName`*	Specify the directory where '.list' files are to be stored. If the directory does not exist or doesn't have the proper permissions, the list file is not generated. This option forces the generation of list files even if -lf was not used.
`-po`	Output full listing to a '.list' file in ANSI fixed format with lines numbering. The list file contains all the source code, all the copybooks are merged into it (unless the SUPPRESS clause is used in the COPY statement) and in most of the cases it can be compiled as it is a standard COBOL program. The line number is printed in the sequence number area (from column 1 to column 6). If used along with -ld, only the source part is generated in ANSI fixed format, the datamap is always in FREE format. This option is guaranteed to work correctly only if the original program is already in ANSI fixed format, otherwise results are unpredictable. **Note:** The listing is generated before the syntax analysis and every dot out of quotes is considered as end of the statement, therefore, if you don't enclose copybook names between quotes, you might obtain an uncompilable list file.

Memory Options

`-m1`	Put all of WORKING-STORAGE into a contiguous block of memory.

Output Options

`-od=`*`DirName`*	Specify the output directory for classes. If the directory does not exist or doesn't have the proper permissions, the compilation will fail.
`-ostrip`	Discard variable names from object files. Variable names are stripped from the compiled object. This option helps save memory and sometimes increases performance, however, exception messages shown by the JVM are less clear and the program cannot be compiled in debug mode.

Perform Stack Options

The -pt options control the behavior of returns from code executed during a PERFORM statement	
`-pt0`	Non-recursive PERFORM, RM/COBOL style
`-pt1`	Recursive PERFORM, Micro Focus COBOL and ACUCOBOL-GT style (default)
`-pt2`	Pseudo non-recursive PERFORM, OS/VS COBOL style

Use -pt0 for compatibility with RM/COBOL, -pt1 for compatibility with the default behavior of Micro Focus COBOL and ACUCOBOL-GT, and -pt2 for compatibility with mainframe behavior of OS/VS COBOL, DOS/VS COBOL, VS COBOL II and COBOL/370.

For more information consult the documentation provided with the specific COBOL dialect

Keywords Options

`-rc=word,customword...`	Change reserved words. Multiple values must be separated by commas. Single words (such as DISPLAY, ACCEPT, or ADD) can be changed, but complex statements (such as READ PREVIOUS or NEXT SENTENCE) cannot. For example, -rc=ACCEPT,GETDATA treats the word GETDATA as ACCEPT. It's not possible to replace a keyword with another keyword using this option. Use -rm if you need to replace a keyword with another keyword.
`-rm=newmeaning,word...`	Change the meaning of reserved words. Multiple values must be separated by commas. This option allows you to deem NULL as LOW-VALUES, COMP-5 as COMP, etcetera. For example, -rm=LOW-VALUES,NULL treats NULL as LOW-VALUES.

`-rw=word...`	Suppress reserved words. Multiple values must be separated by commas. This is useful when one or more keywords are used as item names.
	For example, if the program contains the following variables:
	`77 PRINTER PIC X(32).`
	`77 TAB PIC X(18).`
	Then the following option is necessary in order to compile it correctly: -rw=PRINTER,TAB.

Source Options

`-apost`	Causes figurative constant QUOTE/QUOTES to be evaluated single quotes.
`-big`	Use this option to compile big programs. Several classes are generated. The number of generated classes is conditioned by the iscobol.compiler.max_constants * and iscobol.compiler.max_paragraphs * configuration properties. This option is useful to avoid the "too many constants" error that may appear when compiling huge programs. In rare cases, this option can be used in conjuncion with -sns=Statements in order to get rid of the "code too large" error. Programs compiled with this option are not optimized, so it's suggested to use this option only for those programs that actually require it. You can activate the option using the IMP OPTION Directive in the first line of the source file in order to avoid a dedicated compile command.
`-ce=Ext1...`	Set the default extension for source and copybooks. When it is specified, any source or copy library file name that does not explicitly specify an extension has the default extension appended to it. Multiple values must be separated by the ";" character. On Linux/Unix both ";" and ":" are accepted as separator character.

`-exec=Macro`	Allow the compilation of the specified EXEC macro. *-exec=html* enables the HTML compiler with the following limitations: • if the source code is written in ANSI mode, then the -sa option is mandatory, • the statements EXEC HTML, END-EXEC and COPY "<file name>" must be alone on a single line, • in the HTML code an host variable is a valid COBOL name prefixed by colon, unless the character before the colon is a letter (e.g. parsing the string text:var1, var1 is not considered host variable), • expressions as indexes are not supported as well as the compilation option -cod1, • the listing obtained by the HTML compiler is not compilable.
`-noexec`	Skip EXEC statements.
`-s78c`	Level 78 implies the end of the previous 01 group item.
`-sa`	Force Fixed (aka ANSI) source format. The same effect can be obtained via the SOURCE Directive, specifying >>SOURCE FORMAT FIXED at the top of the source file.
`-sc`	Force all CALL statements to be static. In order to correctly compile a program with this option, all programs called by the program you're compiling must be available in the CLASSPATH. During the runtime session CALLs will perform better, but classes will always be loaded from the CLASSPATH and never from iscobol.code_prefix paths.
`-scnl`	Converts copyfile names to lower case.
`-scnu`	Converts copyfile names to upper case.
`-sdcs`	Allows the currency sign to be changed at runtime. It works in conjunction with the iscobol.runtime.currency * configuration property.
`-sddp`	Allows the DECIMAL-POINT clause to be reverted at runtime. When a program is compiled using this option, the absence or presence of the clause DECIMAL POINT IS COMMA is used only to retrieve the position of the thousands separators and decimal separator inside edited pictures and numeric literals. The actual character used in any display or print statement at runtime is controlled by the configuration property iscobol.runtime.decimal_point_is_comma (boolean) * .

`-sevc`	Supports the syntax:
	`copy "filename" of "$COPYDIR".`
	and
	`copy "$COPYDIR/filename".`
	Environment variables in COPY file names are resolved.

`-sf`	Forces Free source format. The same effect can be obtained via the SOURCE Directive, specifying >>SOURCE FORMAT FREE at the top of the source file.

`-sl`	Allows AREA B to extend to the end of the line, regardless of line length, in Fixed (aka ANSI) format. The same effect can be obtained via the IMP MARGIN-R IS AFTER END OF RECORD Directive, specifying >>IMP MARGIN-R IS AFTER END OF RECORD at the top of the source file.
`-smat`	Allows the mixing of source files and copybooks written in Fixed and Terminal formats. This is useful when you're writing the main source code using the Fixed format and you need to insert some copybooks that are written in Terminal format, or vice versa. The rules that the compiler uses to determine the source format are as follows: The compiler looks at the character in the 7th column of the first not empty line. If this character is a valid indicator (*,$,/) or a blank, then the source file is assumed to be in Fixed (aka ANSI) format. If the first character of the first not empty line is a valid indicator, not including the case of "*>", then the source file is assumed to be in Terminal format. Otherwise it is assumed to be in Free format. As result, a valid comment on the first line establishes the format in a portable way. The -smat option has two effects: 1) If the file is not in Fixed (aka ANSI) format then the file is in Terminal format 2) The analysis is repeated again for each copy file

`-smfu`	Treat hyphen and underscore as different characters in identifiers.
	
	With this option the following code:
	
	`77 data-item-1 pic x(10).`
	`77 data_item_1 pic 9(9).`
	
	doesn't produce an ambiguos identifier error.

`-sns=`*`Statements`*	Use this option to avoid "code too large" compiler errors or to optimize the compiled class file. This option sets the maximum number of COBOL statements that the compiler should attempt to keep together in a single generated Java method. A reasonable value for this option is 200. Try lower values if the "code too large" error persists. In rare cases, the -big option has to be used as well in order to get rid of the "code too large" error. Reason for the "code too large" error: one of the static constraints on the instructions in Java virtual machine code in a class file is that the bytecode size of a single method must be less than 65536 bytes. For COBOL this means that the compiler code generator must split large COBOL paragraphs and sentences into multiple Java methods. See https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-4.html for more information.

`-sp=`*`Copypath...`*	Specify all paths in which COPY files and resources can be found. Multiple values must be separated by the ";" character. On Linux/Unix both ";" and ":" are accepted as separator character. The current directory is always assumed as first path of the *Copypath*. The location of isCOBOL's default copybooks can be specified with this option. If not specified, the Compiler automatically searches for them in the directory "../sample/isdef", resolved from the location of the iscobol.jar library. For more information about isCOBOL's default copybooks see Copybooks.
`-ssnl`	Converts subroutine names to lower case.
`-ssnu`	Converts subroutine names to upper case.
`-st`	Forces Terminal source format. The same effect can be obtained via the SOURCE Directive, specifying >>SOURCE FORMAT TERMINAL at the top of the source file.
`-stl=`*`Length`*	Set the length of a tab character (the default value is 8). Multiple lengths are allowed, for example: "-stl=4,8" sets the first tab to 4 characters in length, and the rest to 8 characters in length.
`-sv`	Forces Variable source format. The same effect can be obtained via the SOURCE Directive, specifying >>SOURCE FORMAT VARIABLE at the top of the source file.

Screen Options

-va	AUTO assumed on all ACCEPT statements.
-vansi	Treat simple ACCEPT and DISPLAY statements in accordance with ANSI semantics. Specifying this option is the same as specifying FROM CONSOLE for all simple ACCEPT statements and UPON CONSOLE for all simple DISPLAY statements. You can change this behavior for individual ACCEPT or DISPLAY statements by specifying an explicit FROM/UPON phrase.
-vh	HIGHLIGHT assumed on all ACCEPT and DISPLAY statements.
-vu	WITH UPDATE assumed on all ACCEPT statements.
-vx	Allows exception keys to be entered by the user for any ACCEPT statement. This option assumes ON EXCEPTION CONTINUE for each ACCEPT that does not specify the ON EXCEPTION clause.

Warning Options

-watn	Show warnings for MOVEs of alphanumeric items to numeric items.
-wd2	Show warnings for features that are currently not supported by WebDirect. This option helps the programmer understand how their program will behave when running with WebDirect.
-wdbz	Show warnings for possible divide by zero without ON SIZE ERROR.
-whttp	Show warnings for statements that are not supported by EIS/Mobile.
-wlu	Show warnings for LINKAGE/USING mismatch, if there are parameters that are defined in the program LINKAGE SECTION but not in the PROCEDURE DIVISION USING phrase.
-wmwc	Show warnings for long variables in MOVE WITH CONVERSION. It affects also the MOVE with conversion from alphanumeric to numeric edited, hence warnings may be shown also for normal MOVE from alphanumeric to numeric edited if -caec is used in addition to this option.
-wr	Extends the REDEFINES TOO LONG warning also to group items. Without this option the warning is returned only when both the redefined item and the redefining item have a picture.
-wref	Show warnings for reference modifiers out of range. **Note:** with this option the error *#173 Reference modifier out of range* is returned as a Warning instead of a Severe error.
-wu	Show warnings for variables that are not used. This option is useful for programmers that wish to perform code cleaning by removing useless variable definitions. Consider that by default, the compiler does not allocate data division items that are not used, so this option is useful only to reduce the number of source lines and not the memory usage.

Miscellaneous Options

`-tasks`	Prints tasks in the compiler output. All comments starting with “todo” are considered tasks.
`-ze`	Automatically execute the program when the compilation is finished.
`-zi`	Set the program to INITIAL. RESIDENT programs are not affected by this option.
`-zmf`	Generates optimized class. Only batch programs without user interface can be compiled with this option. Only ANSI-85 syntax is supported. ESQL is not supported. A new generation engine is used, so the console output could be slightly different. This option can’t be used in the IDE, it’s supported only on the command line. This option can’t be used along with -d and -dx, hence programs compiled with this option can’t be debugged. The activity of programs compiled with this option is not traced in the logfile.
`-zy`	Use 4-digit year in ACCEPT FROM DAY/DATE. Treat ACCEPT FROM DATE as ACCEPT FROM CENTURY-DATE and ACCEPT FROM DAY as ACCEPT FROM CENTURY-DAY

Compiler Properties

The list of configuration properties that affect the Compiler behavior can be found at Compiler Configuration.

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

Compiler Directives

The compiler directives can alter the behavior of the compiler.

When the source format is fixed or terminal, a compiler directive shall be written in the program-text area and may be followed only by space characters and an optional inline comment. When the reference format is free-form, a compiler directive may be followed only by space characters and an optional inline comment. When the alternative syntax is used, a compiler directive can be followed by a dot.

The universal marker for every directory is a pair of greater signs (>>). The directive name and value must be spcified after this marker. The marker can appear in every column of the source line except the sequence number area (columns 1 to 6) of the Fixed source format.

Some directives can also be specified with an alternative marker, supported for compatibility with other COBOLs. Refer to the documentation of each specific directive to know which markers are supported for that directive.

Specific directives are described next.

DOC Directive

The DOC directive allows you to include Javadoc comments into the compiled class.

```
>>DOC
*  Text
*  Text
*  Text
>>END-DOC
```

Or

```
*((DOC))
*  Text
*  Text
*  Text
*((END-DOC))
```

Or

```
*>((DOC))
*> Text
*> Text
*> Text
*>((END-DOC))
```

Or

```
$DOC
*  Text
*  Text
*  Text
$END-DOC
```

Syntax:

1. The directive must appear over a PROGRAM-ID Paragraph, a CLASS-ID Paragraph or a METHOD-ID Paragraph. When used elsewhere it's treated as commentary.
2. The text between DOC and END-DOC must be prefixed by the proper comment marker depending on the source format. The source format also limits the length of each line of text.
3. If END-DOC is omitted, the whole block is treated as commentary.

Example

Describe the scope of a method in a CLASS-ID program:

```
      >>doc
      * The checkEmailAddr method verifies the correctness of an email
      * address. The address must include the @ character and a dot to
      * be considered as valid. The method returns true if the email
      * address is valid or false otherwise.
      >>end-doc
       identification division.
       method-id. checkEmailAddr as "checkEmailAddr".
       ...
```

DEFINE Directive

The DEFINE directive defines a compiler constant.

```
>> DEFINE ConstantName AS ConstantValue [OVERRIDE]
```

Syntax:

1. *ConstantName* is the name of the constant to be set. This name is case insensitive.
2. *ConstantValue* is the value of the constant. If the value is numeric, it can be specified either between quotes or without quotes. If the value is alphanumeric, it must be specified between quotes.

General rules:

1. The Constant is defined and set to the *ConstantValue*.
2. Use the OVERRIDE clause to re-define a previously set constant.

Alternative syntax

The following equivalent syntax is supported for compatibility:

```
$SET CONSTANT ConstantName ConstantValue
```

The dollar sign must appear in the source indicator area.

Example

Define the DEBUG compiler constant set to 1:

```
          >>DEFINE DEBUG AS 1
      program-id. eg001.
        ...
```

EFD Directives

EFD Directives are documented in the Language Reference book.

ELK Directives

ELK Directives are documented in the Language Reference book.

ERROR Directive

When the ERROR directive is encountered, a message is written to STDERR and the compiler exits.

```
>> ERROR String
```

Syntax:

1. *String* is a text string delimited by quotes.

Example

Avoid a useless test program to be compiled:

```
           >>ERROR "test program, not to be used"
       program-id. eg001.
         ...
```

EVALUATE Directive

The EVALUATE directive checks the value of a constant to determine the inclusion of lines of source code.

```
>> EVALUATE ConstantName
  { >> WHEN ConstantValue Statement-1 } ...
  [ >> WHEN OTHER Statement-2 ]
>> END-EVALUATE
```

Syntax:

1. *ConstantName* is a constant defined in the configuration file as iscobol . compiler . const . ConstantName or using the DEFINE Directive compiler directive. This name is case insensitive.
2. *ConstantValue* is the value to be tested. If testing constants with numeric values, *ConstantValue* can be specified either between quotes or without quotes. If testing constants with alphanumeric values, *ConstantValue* must be delimited by quotes.
3. *Statement-1* and *Statement-2* are lines of COBOL code.

General rules:

1. When *ConstantValue* matches the value of *ConstantName*, the lines in *Statement-1* are included.
2. When no match is found, the lines in *Statement-2* are included.

Example

Show a message only if the DEBUG constant is set either to 1 or 2:

```
           >>EVALUATE DEBUG
           >>WHEN 1
           display message "debug level: 1"
           >>WHEN 2
           display message "debug level: 2"
           >>END-EVALUATE
       ...
```

IF Directive

The IF directive checks the value of a constant to determine the inclusion of lines of source code.

```
>> IF ConstantName IS [ NOT ] DEFINED Statement-1
                     {[ NOT ] = ConstantValue}
                     {[ NOT ] < ConstantValue}
                     {[ NOT ] > ConstantValue}
  [ >> ELSE Statement-2 ]
>> END-IF
```

Syntax:

1. The IS word is optional.
2. *ConstantName* is a constant defined in the configuration file as iscobol . compiler . const . ConstantName or using the DEFINE Directive compiler directive. This name is case insensitive. If testing constants with numeric values, *ConstantValue* can be specified either between quotes or without quotes. If testing constants with alphanumeric values, *ConstantValue* must be delimited by quotes.
3. *Statement-1* and *Statement-2* are lines of COBOL code.

General rules:

1. Testing IF DEFINED, when *ConstantName* is defined, no matter the value, the lines in *Statement-1* are included. Otherwise, the lines in *Statement-2* are included.
2. Testing IF EQUAL, GREATER or LESS, when *ConstantName* is defined, and its value matches the IF condition, the lines in *Statement-1* are included. Otherwise, the lines in *Statement-2* are included.

Alternative syntax

The following equivalent syntax is supported for compatibility:

```
$IF ConstantName [ NOT ] DEFINED Statement-1
                {[ NOT ] = ConstantValue}
                {[ NOT ] < ConstantValue}
                {[ NOT ] > ConstantValue}
  [ $ELSE Statement-2 ]
$END
```

The dollar sign must appear in the source indicator area.

Example

Show a message only if the DEBUG constant is defined and set to 1:

```
           >>IF DEBUG IS DEFINED
           >>IF DEBUG = 1
           display message "debug mode"
           >>END-IF
           >>END-IF
       ...
```

IMP MARGIN-R IS AFTER END OF RECORD Directive

The `IMP MARGIN-R IS AFTER END OF RECORD` directive allows AREA B to extend to the end of the line, regardless of line length, in Fixed (aka ANSI) format. The same effect can be obtained via the -sl compiler option.

```
>> IMP MARGIN-R IS AFTER END OF RECORD  String
```

Example

In the following program it is possible to write code after column 72:

```
           >>IMP MARGIN-R IS AFTER END OF RECORD
       program-id. eg001.
         ...
```

IMP OPTION Directive

The `IMP OPTION` directive sets compiler options for a program.

```
>> IMP OPTION String
```

Syntax:

1. *String* is a text string delimited by quotes that contains the compiler options. Refer to the Compiler Options section for further details.
2. This directive must appear as first row in the source file. It cannot be used in the body of the source code.

General rules:

1. If the character "-" precedes the option, it's used during the compilation process; if the character "#" precedes the option, it is *not* used during the compilation process.
2. Options with a value override the value specified on the command line, if any. For example, if you have -sns=100 in the command-line and "-sns=200" in the IMP OPTION directive, the program will be compiled with sns set to 200.
   In order to remove an option with a value, the equal sign must be specified along with the option name. For example, if you have -sns=100 in the command-line and you want to remove it through IMP OPTION, the correct syntax is >>IMP OPTION "#sns=" not just >>IMP OPTION "#sns".
3. The following compiler options can be used only in the command line and are ignored if they appear in *String*: -exec, -la, -lf, -lfo, -lo, -od, sf, -st, -sa, -sl, -sv, -smat and -zmf.
   When these options are used in *String* they don't produce any effect but they are still registered in the list of used options and so displayed by the -info runtime option.

Example

The following program will be always compiled with -dz and never compiled with -di:

```
           >>IMP OPTION "-dz #di"
       program-id. eg001.
         ...
```

PROPAGATE Directive

The PROPAGATE directive activates or deactivates propagation of exceptions for CALL statements.

```
>> PROPAGATE { ON  }
             { OFF }
```

General rules:

1. ON activates propagation, OFF deactivates it. The default value is OFF.

   When propagation is OFF, and a CALL statement without the ON EXCEPTION clause fails, an error is shown and execution terminates.

   When propagation is ON and a CALL statement without the ON EXCEPTION clause fails, the COBOL program terminates and reports the exception to the caller. The Runtime Framework keeps reporting the same exception to the caller until a program without propagation, or a CALL statement with an ON EXCEPTION clause is found.

   ### Example

Propagation will be activated in the following program:

```
          >>PROPAGATE ON
      program-id. eg001.
        ...
```

SET Directive

The SET directive allows you to set a Compiler Configuration property for a program.

```
>> SET [NO] PropertyName PropertyValue
```

Syntax:

1. *PropertyName* is a text string that specifies the property name. It can be delimited by quotes.
2. *PropertyValue* is a text string that specifies the property value. It can be delimited by quotes. It can be written between parenthesis.

General rules:

1. *PropertyName* can be any of the properties described in Compiler Configuration stripped of the "iscobol.compiler" prefix.
   Only underscore is allowed as separator between words. Replacing underscore with hyphen invalidates the directive name.
   PropertyName is case insensitive.
2. The following properties cannot be set:

- custompreproc
- max_paragraphs
- max_constants
- max_hostvars
- options

- regexp

3. When the NO keyword is used, then *PropertyValue* shouldn't be specified. In this way, the property is unset.

Alternative syntax

The following equivalent syntax is supported for compatibility:

```
$SET [NO] PropertyName PropertyValue
```

The dollar sign must appear in the source indicator area.

Example

A Java bridge program will be generated for the following program:

```
           >>SET "easylinkage" "1"
       program-id. eg001.
         ...
```

SOURCE Directive

The SOURCE directive sets the format of the source code.

```
>> SOURCE FORMAT IS { FIXED    }
                    { FREE     }
                    { PREVIOUS }
                    { TERMINAL }
                    { VARIABLE }
```

Syntax:

1. The IS word is optional.
2. FIXED is ANSI format.
3. FREE is Free format.
4. PREVIOUS restores the format that was set before the last SOURCE directive.
5. TERMINAL is Terminal format.
6. VARIABLE is Variable format.

General rules:

1. The SOURCE FORMAT directive indicates that the source text or library text following the directive and continuing through a subsequent SOURCE FORMAT directive shall be treated as fixed form if FIXED is specified, as terminal form if TERMINAL is specified, as variable form if VARIABLE is specified or as free form if FREE is specified. See Source Formats for more information about the different source formats.
2. The default reference format of a compilation group is fixed form.
3. The default reference format of library text is the reference format that was in effect for the COPY statement that resulted in processing of this library text.
4. If a SOURCE FORMAT directive is specified in library text, the specified format shall be in effect until another SOURCE FORMAT directive is encountered or the end of the library text is reached. When the processing of that library text is completed, the reference format shall revert to the reference format that was in effect for the COPY statement that resulted in processing of that library text.

5. This directive overrides -sa, -sf, -smat, -st and -sv compiler options.

Example

The source code of the following program will be treated according to the FREE format rules:

```
      >>SOURCE FORMAT FREE
program-id. eg001.
 ...
```

SQL Directives

SQL Directives are documented in the Language Reference book.

Source code preprocessing

The isCOBOL Compiler offers the ability to preprocess COBOL source files either by providing replacement rules via regular expressions or by using preprocessor programs. In both cases the original source files remain unchanged, but the output files generated by the compiler (e.g. the source listing and the compiled class) will be affected by the modifications applied by the source code preprocessing.

Preprocessing the code via regular expressions is the easiest approach but it's limited to the line level. It's just a find-and-replace that the Compiler performs on each source line before parsing it. All the words of the searched string must appear on the same source line in order to be found. For more information on this approach, see Source code preprocessing via regular expressions.

Preprocessing the code via preprocessor programs is a little trickier but it provides more possibilities. Although the line level approach is still used, the preprocessor program can keep track of previously processed lines so it can find text distributed on multiple lines and perform operations that are related to the context. For more information on this approach, see Preprocessor programs.

Source code preprocessing via regular expressions

The isCOBOL Compiler offers the ability to preprocess COBOL source files through regular expressions.

This feature is activated by the configuration property iscobol.compiler.regexp *. If the property is not set, then no preprocessing is performed. If the property is set, then the source code is preprocessed at compile time.

The property iscobol.compiler.regexp * can be set to one or more pairs of strings enclosed between double quotes, in the format:

```
iscobol.compiler.regexp= "match" "replacement" "match" "replacement" ...
```

Each pair of strings identifies a replacement that will be applied to the source before the Compiler processes it.

The first string of each pair is a match string and must respect Java specifications (https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html).

Note - The backslash character (\), if used, must be doubled due to the property file format.

The second string of each pair is the replacement string. Every time the first string matches it is replaced by the second string.

Regular expressions are resolved sequentially in the order they appear in the iscobol.compiler.regexp * value for every line of code. More than one replacement might be performed on the same line; for example, having the following setting:

```
iscobol.compiler.regexp="A.A" "ABA" "B.B" "BCD"
```

the text AXAB will be transformed in ABCD (ABAB after the first replacement and ABCD after the second replacement, that is applied on the result of the first).

Replacements affect all the text in the source code, including string literals.

The list file produced by the Compiler includes the result of the text replacement (see Listing Options for information about how to obtain a list fie).

Examples

Example - The following regexp replaces DISPLAY UPON SYSERR statements with calls to the C$WRITELOG routine, without case sensivity and regardless of how many spaces are between words.

```
iscobol.compiler.regexp="(?i)(DISPLAY)(.+)\\s+(UPON)\\s+(SYSERR)" "CALL 'C\\$WRITELOG'
USING \$2"
```

Preprocessor programs

The isCOBOL Compiler allows you to provide preprocessor programs that will be invoked for each processed source line. These programs must implement the *com.iscobol.compiler.custpreproc.LinePreProcessor* interface so they can be either Java classes or CLASS-ID COBOL programs.

Preprocessor programs are specified through the iscobol.compiler.custompreproc configuration property. For example, with the following configuration

```
iscobol.compiler.custompreproc=PreProc01
```

the following procedure is performed for each line of code:

1. the Compiler reads the line of code,
2. the Compiler invokes PreProc01 passing the line of code to it along with complementary information (i.e. current source format and compiler options),
3. PreProc01 parses the line and optionally changes the content generating one or more different lines of code,
4. the Compiler receives the new content from PreProc01 and parses it.

If more than one preprocessor is specified, e.g.

```
iscobol.compiler.custompreproc=PreProc01 PreProc02
```

then the following procedure is performed for each line of code:

1. the Compiler reads the line of code,
2. the Compiler invokes PreProc01 passing the line of code to it along with complementary information (i.e. current source format and compiler options),
3. PreProc01 parses the line and optionally changes the content generating one or more different lines of code,

4. the Compiler receives the new content from PreProc01 and passes it to PreProc02 along with complementary information (i.e. current source format and compiler options),
5. PreProc02 parses the content and optionally changes it generating one or more different lines of code,
6. the Compiler receives the new content from PreProc02 and parses it.

The preprocessor activity is transparent for the user that launches the Compiler. The original source code remains unchanged. Also during debug, the Debugger shows the original source code.

Implementation

The *com.iscobol.compiler.custpreproc.LinePreProcessor*, that preprocessor programs must implement, has only one method:

```
public void process(java.lang.String originalLine,
                    int sourceFormat,
                    java.lang.String fileName,
                    int lineNumber,
                    java.lang.String[] compilerOptions,
                    com.iscobol.compiler.custpreproc.ProcessResult result)
             throws com.iscobol.compiler.custpreproc.ProcessException
```

Where:

- *originalLine* includes the text read from the source file
- *sourceFormat* tells the current source format:

Value	Meaning
0	Unknown
1	Fixed
2	Terminal
3	Free
4	Variable
5	Long line

Note - unless a specific source format has been forced via compiler options, the source format of the first line will be unknown.

- *fileName* holds the current source file; it could be either the main cbl or a copybook
- *lineNumber* holds the line number in the current source file. If zero, it means that the end of file has been reached and *originalLine* is null.
- *compilerOptions* includes all the compiler options that are in use
- *result* is a *com.iscobol.compiler.custpreproc.ProcessResult* object and contains the following methods:

```
public java.lang.String getReplace()
public void setReplace(java.lang.String replace)
public boolean isComment()
public void setComment(boolean comment)
```

Where:

- *replace* initially matches with *originalLine*, but the preprocessor program can change it. You can set this String either to one line of code or to multiple lines of code. In order to specify a new line of code, use line feed character (x"0A") followed by the proper number of spaces (depending on the source format) and the text of the new line.
  Note - if your preprocessor program is a CLASS-ID, ensure you have set iscobol.compiler.oop.trim_parameters to false in the Compiler's configuration before compiling it, otherwise the spaces you used for indentation might be trimmed.
- *comment* specifies if the text hosted by the replace String should be considered as comment or not. Set comment to true to mark the text as comment.

When the *process* method is completed, the Compiler receives the test hosted by *replace* and the comment switch hosted by *comment*.

The preprocessor program can optionally raise errors and warnings by throwing a *com.iscobol.compiler.custpreproc.ProcessException*. This class has the following methods:

```
public ProcessException(int severity, java.lang.String message)
public ProcessException(int severity, String message,
com.iscobol.compiler.custpreproc.ProcessResult replace)
```

Where:

- *severity* specifies the severity:

Value	Meaning
1	Informational
2	Warning
3	Error
4	Severe error

- *message* includes the error text

- *replace* includes the new source code to be parsed by the Compiler.

Use the second signature when the error is not severe and the compilation process must continue.

Bridge Programs

The source code of bridge programs that are automatically generated due to the iscobol.compiler.easydb (boolean) and iscobol.compiler.servicebridge (boolean) configuration properties is passed to the preprocessor program as well.

Troubleshooting

There are two methods for troubleshooting issues on preprocessor programs:

- add the -lf option to your compile command and look at the list file. The list file contains the new code as it was altered by preprocessors.
- attach a debugger to the preprocessor programs and debug their logic.
  If the preprocessor programs are CLASS-ID COBOL programs, you can proceed as follows:

  a. Compile the preprocessor programs either with -d option or -dx option
  b. Add the *iscobol.compiler.rundebug* setting to your compiler command line, e.g.

```
iscc -J-Discobol.compiler.rundebug=2 ...
```

  c. Launch the remote debugger:

```
iscrun -r
```

Javadoc

Consult the javadoc installed with isCOBOL in the folder $ISCOBOL_HOME/javadoc for the full reference of the com.iscobol.compiler.custpreproc classes.

Examples

Some preprocessor examples are installed with the isCOBOL SDK under the *sample/compiler-pre-process* directory.

External File Description dictionaries

The isCOBOL Compiler is capable of generating data dictionaries that store a map of the COBOL record structures. These dictionaries are called External File Description (EFD) because they're based on the standard COBOL file description (FD).

External File Description dictionaries are used by all the external tools that otherwise would not be aware of the COBOL record structure, including but not limited to the DatabaseBridge generator (edbiis), DCI and GIFE (Index and Relative File Editor).

By default External File Description dictionaries are given the same name of the COBOL file that they describe. If the file name is variable, the name of the variable is used. The name can be customized via the FILE Directive.

External File Description dictionaries have XML content. Each dictionary is divided in three areas:

key area	In this area, keys are described. You can find information such as the number of segments, the name of the fields included in each key and the offset and length of each segment.
conditions area	This area appears only if at least one WHEN Directive was used. It includes the details about the conditions specified by WHEN directives.
fields area	In this area, record descriptions are provided. For each field the following information is provided: name, type, size, length and offset.

The main element of External File Description dictionaries is named "table" and provides the following information: name of the file, record size and number of keys.

The fields of the COBOL structure are renamed as follows in the dictionary:

- all names are made upper-case unless a diffrent case is specified by the NAME Directive,
- all dashes are translated to underscore,
- if the field name begins with a number, an underscore is put before it.

All the fields of the COBOL structure are described in the dictionary, but some of them as marked as hidden, instructing the external tools to ignore them. The following fields are usually marked as hidden:

- fields that have no picture as they have children items, unless the USE GROUP Directive is specified above them,
- fields that are REDEFINES of other fields, unless these fields are part of a key, or a WHEN Directive is specified above them,
- fields of alternate record definitions, unless mapped to different tables via WHEN Directive,
- fields that were hidden on purpose via the HIDDEN Directive.

The isCOBOL Framework includes and internal object, the efdParser Class (com.iscobol.lib.efdParser), that allows you to easily retrieve information from External File Description dictionaries.

Runtime Framework

Overview

Source code compiled with isCOBOL needs a Runtime Framework to run. The Runtime is the engine that runs the application. The Runtime Framework consists of a number of libraries that provide all the functionality required to run the application. There are two kinds of library: Java jar libraries and operating system native libraries.

This is the list of the Java jar libraries that compose the isCOBOL Runtime Framework:

Library	Description
asm-7.2.jar asm-commons-7.2.jar asm-tree-7.2.jar	Bytecode manipulation framework. These libraries are required for the Code Coverage feature.
bcprov-jdk14-1.38.jar	Bouncy Castle library used for PDF encryption. If your programs don't encrypt PDF files using either the ENCRYPTION attribute or the iscobol.print.attribute.encryption configuration property, you could safely remove this library.
charva.jar	Support for "green" terminals.
commons-codec-1.13.jar commons-collections4.4.4.jar commons-compress-1.19.jar commons-math3-3.6.1.jar poi-4.1.2.jar poi-ooxml-4.1.2.jar poi-ooxml-schemas-4.1.2.jar xmlbeans-3.1.0.jar	These libraries provide the ability to manage XLS and XLSX files. They're required by the Grid control export features.
commons-logging-api.jar	Implementations of commons-logging wrapper API.
commons-logging.jar	Implementatinos of commons-logging.
ctree-rtg.jar	ctreej interface.
DJNativeSwing-SWT.jar DJNativeSwing.jar swt-<platform>.<bits>.jar	Default web-browser control implementation.
image4j-0.7.2.jar	This library provides support for 1 and 32 bits .bmp files.
iscobol.jar	isCOBOL Compiler, Framework, Debugger and Application Server.
isupdater.jar	isCOBOL Software Updater tool.
isprofiler.jar	isCOBOL Profiler.
itext-2.1.7v5.jar	This library allows you to generate PDFs for print files assigned to "-P PDF".
jacoco-core-0.8.5.jar	This library is required for Code Coverage and Profiler features.
javassist.jar	isCOBOL Profiler dependences.
jcalendar-1.3.3.jar	Date-entry control implementation.
jcommon-1.0.23.jar jcommon-xml-1.0.23.jar jfreechart-1.5.1.jar	This library allows you to create charts via java-bean technology.
jdom.jar	This library allows the COBFILEIO, EfdParser and XML2WRK to parse XML files.

Library	Description
jna-5.2.0.jar jna-platform-5.2.0.jar	These libraries allow the C$SYSTEM routine to create a process on Windows. They also allow the C$OPENSAVEBOX routine to display native file chooser dialogs on Windows. They're required also by the default web-browser control implementation.
joe-1.3.jar	This library allows you to execute joe scripts.
wow.jar wowax.jar	isCOBOL WOW support.

These libraries are installed in the isCOBOL lib folder on all platforms. All these libraries are portable to different platforms except for "swt-<platform>.<bits>.jar" that includes native items and therefore is different on every operating system.

This is the list of native libraries included in the isCOBOL Runtime Framework:

Library	Description
ctree	allow to interact with a c-tree server
dyncall	allows programs compiled without -cp option to call dynamic link libraries
dyncall_n	allows programs compiled with -cp option to call dynamic link libraries
iscobolc	allows programs compiled without -cp option to be called by C programs
iscobolc_n	allows programs compiled with -cp option to be called by C programs
Terminal	Curses implementation (native part) of Charva. Note - This library is not available on the Windows 64 bit platform.

On Windows systems these libraries are installed in the isCOBOL bin folder and have dll extension (e.g. bin\dyncall.dll).

On Linux/Unix systems these libraries are installed in the isCOBOL native/lib folder, they have the lib prefix and so extension (e.g. native/lib/libdyncall.so).

On Mac OSX systems these libraries are installed in the isCOBOL native/lib folder, they have the lib prefix and jnilib extension (e.g. native/lib/libdyncall.jnilib).

A program compiled with isCOBOL can be executed with the following command:

```
iscrun ProgramName
```

Note - On Windows this command should be launched from inside the isCOBOL Shell. Otherwise you need to set ISCOBOL and ISCOBOL_JRE_ROOT environment variables before using iscrun.

This command is a wrapper which automatically adds all of the JAR files listed above to the class path before executing java and passing *ProgramName* to it. When running on Windows, the following command can be used in the same way to call javaw.exe:

```
isrun ProgramName
```

Javaw.exe runs the program without displaying the command line console window.

When using isrun.exe, since standard output and standard error are not available in this case, the console output printed on two files called "isrun_out.log" and “isrun_err.log” in the bin directory of isCOBOL.

Paths in *Program-Name* are considered only if iscobol.code_prefix is set. Relative paths in *Program-Name* are appended to the code-prefix paths.

Runtime Options

The Runtime Framework has a number of properties which can affect the behavior of isCOBOL. These allow the user to specify runtime actions such as authenticating passwords, customizing remote debugging, and specifying file systems. Properties can be set in the configuration file or dynamically changed by the application. A complete list of properties and their definitions is located in the Configuration section below. Here are a couple of examples of commonly used options.

-c

The -c option allows you to pass an additional configuration file:

```
iscrun -c myApp.cfg ProgramName
```

The properties found in the configuration file are appended to the existing configuration.

-coverage

The -coverage option generates a report of the code actually executed in the runtime session. See isCOBOL Code Coverage for more information.

```
iscrun -coverage ProgramName
```

-d

The -d option runs the program in debug mode:

```
iscrun -d ProgramName
```

See Debugger for more information about debugging.

-info

The -info option displays information about the program class file. For example:

```
iscrun -info ProgramName
```

The returned information contains:

- the class location
- the list of options that were used to compile the program,
- the version of the Compiler (build number) that compiled the program,
- the minimum runtime version (build number) necessary to run the program,
- the list of resources that were embedded via COPY RESOURCE statements.

-iut

The -iut option activates the Unit Test feature, allowing you to check if the programs are behaving as expected. It generates a report of the result at the end of the run unit. See isCOBOL Unit Test for more information.

-joe

The -joe option allows you to start JOE's CobShell. There are two usages:

- open the CobShell in interactive mode:

```
iscrun -joe
```

- run a script:

```
iscrun -joe ScriptName.joe
```

See JOE for more details.

-license

The -license option displays information about the isCOBOL license.

```
iscrun -license
```

It's good practice to use this option along with -c, if applicable, in order to have the runtime looking in every possible configuration to find the active license code. For example:

```
iscrun -c myApp.cfg -license
```

-profile

The -profile option profiles the runtime activity and generate a report:

```
iscrun -profile ProgramName
```

See Profiling COBOL programs for details.

-t

The -t option runs in terminal mode:

```
iscrun -t ProgramName
```

See Using CHARVA for details.

-update

Using the -update option causes the runtime to look for updates through ISUPDATER (Update Facility) before starting.

```
iscrun -update [-uc updater-config] ProgramName
```

The -uc option allows you to specify a custom isUpdater configuration file. If the -uc option is not used, the runtime looks for a file named *isupdater.properties* in the Classpath.

The configuration file must contain the setting to connect to a update server through HTTP. This setting is swupdater.site. The HTTP server should be properly configured to provide an update of the isCOBOL SDK. See Setup of an update server for the isCOBOL SDK for details.

The configuration file could contain also additional settings (see Update Facility's Client Configuration (isupdater.properties) for details).

If the configuration file includes only swupdater.site, the runtime uses a default configuration built according to the isCOBOL installation directory on the local PC, for example it sets *swupdater.version.iscobol* to the build number of *lib/iscobol.jar*, *swupdater.directory.iscobol* to the path of the *lib* folder and *swupdater.directory.iscobolNative* to the location of native libraries (*bin* folder under Windows, *native/lib* folder on other platforms).

The need of updating is determined by comparing the build numbers specified by the swupdater.version properties used by the runtime with the build numbers specified by the swupdater.version properties in the server side *swupdater.properties* file.

a. If the server is down or no update is necessary, the runtime execution continues normally
b. If some updates were executed, the runtime is automatically restarted.

-utility

The -utility option allows you to run a utility:

```
iscrun -utility UtilityName
```

UtilityName can be any of the following (case insensitive):

- cobfileio
- cpk
- gife
- isl
- ismigrate
- jdbc2fd
- jutil
- xml2wrk

-v

The -v option displays version information about isCOBOL.

```
iscrun -v
```

-vv

The -vv option displays version information about isCOBOL and the current file handler (the file handler specified by the iscobol.file.index configuration property, whose default value is "jisam").

```
iscrun -vv
```

--system

The --system option forces the current system Look and Feel for the GUI (default if none of the next four options is used):

```
iscrun --system ProgramName
```

--metal

The --metal option forces the Metal Look and Feel for the GUI:

```
iscrun --metal ProgramName
```

--motif

The --motif option forces the Motif Look and Feel for the GUI:

```
iscrun --motif ProgramName
```

--GTK

The --GTK option forces the GTK Look and Feel for the GUI (not available on Windows):

```
iscrun --GTK ProgramName
```

--nimbus

The --nimbus option forces the Nimbus Look and Feel for the GUI:

```
iscrun --nimbus ProgramName
```

Multiple options on the command-line

All the above options except for -v and -info can be combined and appear in any order in the command-line. For example, if you want to debug your program using a specific configuration file and forcing the Motif Look and Feel, you use:

```
iscrun --motif -d -c myApp.cfg ProgramName
```

It's not possible to use -coverage and -profile together.

Configuration

The behavior of Compiler, Runtime Framework and utilities can be customized with properties defined in a configuration file, that contains a list of properties consisting of the name of the property and its value, separated by an equals symbol (=), a colon (:) or a space:

```
Property=Value
Property:Value
Property Value
```

Property may contain the following characters only: lower-case letters (unless specified otherwise in this document), digits, periods and underscores to separate words Spaces, hyphens and any other characters are not allowed. The prefix "iscobol." is mandatory for any property, unless they're set as system environment variables.

Value is trimmed. Leading and trailing spaces are removed. If you need to keep leading spaces, then start with a "\" (backslash).

The backslash character can be used to

- preserve leading spaces in the property value, e.g.

```
myproperty=\    this is the value
```

- split the property value on multiple lines, e.g.

```
myproperty=this \
           is \
           the \
           value
```

- escape characters, e.g.

```
myproperty=C:\\folder1\\folder2
```

Common escape sequences:

Escape sequence	Meaning
\\	backslash
\f	form feed (0x0C)
\n	line feed (0x0A)
\r	carriage return (0x0D)
\u####	unicode representation (#### are the hex representation of the digit in UTF-16 BE encoding)

The property value can include delimited free text (comment) or Java properties values. See iscobol.conf.var_delimiters for more details.

Unless iscobol.conf.only * is set (or the corresponding command line option *-conly* is used, where applicable), isCOBOL can use several configurations at the same time. They are loaded in the following order, and the properties they contain are set in the order in which they appear in the respective files:

1. system environment variables
   Note - isCOBOL only looks for system environment variables whose name is in upper case. This is not a problem on case insensitive systems like Windows. On case sensitive systems like Linux, instead, the system environment variable will not be found if its name is in lower or mixed case.
   Before looking for a configuration property among system environment variables, the Runtime automatically converts the property name to upper-case and replaces any dot with underscore.
2. /etc/iscobol.properties;
3. iscobol.properties located in the user home directory;

4. iscobol.properties located in the Java Classpath;
5. -Discobol.conf=*ownconfigurationfile* or, where applicable, -c *ownconfiguration file* set at the command line;
6. $ISCOBOL/iscobol.properties
7. -Discobol.* (set at the command line).
8. -Discobol.remote_conf=*remoteconfigurationfile* and -c *remoteconfigurationfile* (set at the client command line in Thin Client environment)

in addition to these configuration files, if iscobol.properties is located in a non-standard directory, you can also specify

```
-Discobol.conf=class_name
```

to indicate the proper location. In this example, iscobol.properties is loaded in the same directory as class_name. The class_name needs to be loaded from a .jar file or from a folder (though typically with a package) which also contains iscobol.properties. This jar file or folder must be found in CLASSPATH.

You may also use a configuration file located on a web site by specifying:

```
-Discobol.conf=http://www.yourwebsite.com/iscobol.properties
```

When a property is set more than one time, the most recently set value is used. The properties appearing in the command line have the highest priority, overriding any other previously set value.

For boolean properties, only the first character is used. Therefore, a boolean property is True when its first character is "T", "t", "Y", "y", or a digit from "1" to "9". Any other value is False.

The value of a property can be retrieved in a COBOL program with the following statement:

```
ACCEPT PropertyValue FROM ENVIRONMENT "PropertyName"
```

The value of a property can be temporarily set, for the duration of the working session, with the following statement:

```
SET ENVIRONMENT "PropertyName" TO PropertyValue
```

PropertyName should not begin with "iscobol.". That prefix is automatically added by the Runtime.

PropertyName is case insensitive. The Runtime normalizes it before searching for it in the environment.

The value of a property can also be retrieved via the C$GETENV library routine and can be set via the C$SETENV library routine. These routines might be useful in a thin client environment as they allow you to manage the client-side environment if you call them with a CALL CLIENT statement.

Configuration Properties

The following tables list all the available configuration properties.

- Licenses Configuration
- Compiler Configuration
- General Runtime Configuration
- RemoteCompiler Configuration

- WebDirect Configuration
- HTTPHandler Configuration
- User Interface Configuration
- Debugger Configuration
- File Handling Configuration
- DatabaseBridge and JDBC/ESQL Configuration
- isCOBOL Server (thin client) Configuration
- LoadBalancer Configuration
- Print Configuration
- IDE Reports Export feature Configuration
- Update Facility Configuration
- Library Routines Configuration
- Keyboard Configuration

Licenses Configuration

Licenses properties cannot be set by SET ENVIRONMENT within the program. They must appear in the external configuration.

Property	Meaning
`iscobol.compiler.license.2023`	This property specifies the license code for the isCOBOL Compiler.
`iscobol.easydb.license.2023`	This property specifies the license code for isCOBOL DatabaseBridge.
`iscobol.eis.license.2023`	This property specifies the license code for isCOBOL EIS.
`iscobol.license.2023`	This property specifies the isCOBOL license code. This license activates runtime and debugging features and controls how many concurrent users can connect to the Application Server.
`iscobol.mobile.license.2023`	This property specifies the license code for isCOBOL Mobile.
`iscobol.udbc.license.2023`	This property specifies the license code for isCOBOL UDBC.

Compiler Configuration

Unlike Framework properties, Compiler properties cannot be set by SET ENVIRONMENT within the program. Most of them can be set within the program using SET Directive if stripped of the "iscobol.compiler" prefix.

(*) The asterisk after the property name means that the property can't be set using SET Directive, but can only appear in the external configuration.

Property	Meaning

`iscobol.compiler.command_line_linkage (boolean)`	**True** = command line parameters are moved to the Linkage Section
	False = command line parameters can't be intercepted via Linkage Section.
	
	When set to true, the command line is passed to the main program using the following parameter structure:
	
	`LINKAGE SECTION.`
	`01 CmdLine.`
	`  03 CmdLineLen  pic 9(4)comp.`
	`  03 CmddLineData pic x(256).`
	
	The maximum allowed command line length is 2048 characters. Longer command line values will be truncated.
	
	The default vale is False.

`iscobol.compiler.const.`*`ConstantName`*	Sets the value for a constant that can be used by the compiler directives. *ConstantName* is case insensitive but, since this is a configuration property, it's good practice to use lower case text. The characters in the value define the constant type. A value of characters between 0 and 9 (included) defines a numeric constant. A value that includes any other character defines an alphanumeric constant. Refer to the Compiler Directives section for further details.
`iscobol.compiler.customppreproc`	Specifies one or more preprocessor programs. Program names must be separated by space. See Preprocessor programs for more information about preprocessors.
`iscobol.compiler.efd_field_name_num (boolean)`	**True** = field names that begin with a number are registered as they are in the EFD/ISS dictionaries. **False** = field names that begin with a number are registered with a leading underscore in the EFD/ISS dictionaries. This option affects the generation of EFD/ISS dictionaries as well as the generation of EDBI routines performed when iscobol.compiler.easydb (boolean) is set to true. If omitted, *False* is assumed.

Property	Meaning

`iscobol.compiler.esql.array.`*`TypeName`*	This property specifies the field type within a ARRAY SQL type.
	It's used in conjunction with iscobol.compiler.esql.procedure.ProcedureName or the HOSTVAR Directive when they define parameters with *dbtype*=ARRAY.
	
	TypeName must be written in lower case regardless of the case it has on the database.
	
	For example, given the following procedure, that returns an array of varchar fields:
	
	`create or replace package MYPKG`
	`IS`
	`  TYPE Tbl_TYPE`
	`  IS`
	`     TABLE OF VARCHAR(8)`
	`     INDEX BY BINARY_INTEGER;`
	
	`PROCEDURE GET(P1 OUT Tbl_TYPE);`
	`END MYPKG;`
	
	You would set:
	
	`iscobol.compiler.esql.procedure.mypkg.get=o,array,tbl_type`
	`iscobol.compiler.esql.array.tbl_type=varchar`

`iscobol.compiler.esql.db2 (boolean)`	This property is deprecated and supported only for backward compatibility. Use the -csdb2 compiler option instead. **True** = manage ESQL in compatibility with IBM DB2. It activates the support for the SQLDA structure and a peculiar handling of DATE, TIME and TIMESTAMP parameters in functions. It also allows you to intercept the result of a function or a special register with a SET statement. **False** = manage ESQL in the standard way. If omitted, *False* is assumed.

Property	Meaning
`iscobol.compiler.esql.db2.fun.`*`FunctionName`*	When the -csdb2 compiler option is used, the following DB2 functions are intercepted and handled by the Compiler: ADD_DAYS, ADD_HOURS, ADD_MINUTES, ADD_MONTHS, ADD_SECONDS, ADD_YEARS, AGE, DATE_PART, DATE_TRUNC, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFWEEK_ISO, DAYOFYEAR, DAYS, DAYS_BETWEEN, DAYS_TO_END_OF_MONTH, DATE, EXTRACT, FIRST_DAY, ROM_UTC_TIMESTAMP, HOUR, HOURS_BETWEEN, JULIAN_DATE, MICROSECOND, MIDNIGHT_SECONDS, MINUTE, MINUTES_BETWEEN, MONTH, MONTHNAME, MONTHS_BETWEEN, NEXT_DAY, NEXT_MONTH, NEXT_QUARTER, NEXT_WEEK, NEXT_YEAR, QUARTER, ROUND, ROUND_TIMESTAMP, SECOND, SECONDS_BETWEEN, THIS_MONTH, THIS_QUARTER, THIS_WEEK, THIS_YEAR, TIME, TIMESTAMP, TIMESTAMP_FORMAT, TIMESTAMP_ISO, TIMESTAMPDIFF, TIMEZONE, TO_CHAR, VARCHAR_FORMAT, WEEK, WEEK_ISO, WEEKS_BETWEEN, YEAR, YEARS_BETWEEN, YMD_BETWEEN If programs use other functions (e.g. user defined functions) that require DATE, TIME or TIMESTAMP parameters, you can inform the Compiler through this configuration property, that should be set for each function with DATE, TIME or TIMESTAMP parameters not listed above. The value of this property is the list of database types for each parameter. Semicolon is used as separator between multiple database types. Any Java SQL type is allowed. Refer to the java.sqlTypes javadoc for the list of possible values. For example, assuming that the programs invoke an user defined function named MY_TS_BUILDER that takes a DATE parameter, a TIME parameter and a BOOLEAN parameter and returns a TIMESTAMP as result, you should set: `iscobol.compiler.esql.db2.fun.my_ts_builder=DATE;TIME;BOOLEAN`

Property	Meaning

`iscobol.compiler.esql.procedure.`*`ProcedureName`*	This property specifies the signature of a stored procedure. The compiler uses this information to bind the host variables used as parameters. The value of this property has the following format:
	
	type[,dbtype[,typename]];type[,dbtype[,typename]];...;type[,dbtype[,typename]]
	
	Where:
	• *type* is the parameter type; use "i" for input, "o" for output and "u" for input-output.
	• *dbtype* is the type of the corresponding database field. Any Java SQL type is allowed. Refer to the java.sqlTypes javadoc for the list of possible values. If the *dbtype* is ARRAY, specify also the type name; the Compiler will look for the configuration property iscobol.compiler.esql.array.TypeName in order to know which field type is used in the array. If *dbtype* is omitted, *dbtype*=OTHER is assumed.
	
	ProcedureName must be written in lower case regardless of the case it has on the database. If the procedure is included in a package, the package name must be specified as well.
	
	For example, given the following procedure:
	
	`create or replace package MYPKG`
	`IS`
	`PROCEDURE PROC1( P1 IN NUMBER, P2 OUT VARCHAR);`
	`END MYPKG;`
	
	You would set:
	
	`iscobol.compiler.esql.procedure.mypkg.proc1=i,numeric;o,varchar`

`iscobol.compiler.generate.keep_structure (boolean)`	**True** = Recreate the "bean", "easydb", and "servicebridge" subfolders in the output folder pointed by the -od=DirName option when compiling DatabaseBridge routines and ServiceBridge programs. **False** = Put the classes in the output folder pointed by -od=DirName option when compiling DatabaseBridge routines and ServiceBridge programs. These classes will be mixed with the classes of other COBOL programs. The default value is False. **Note -** the "easylinkage" folder is not affected by this setting.
`iscobol.compiler.generate.root_dir`	Specifies the directory where the Compiler will create the subfolders to host the source code for DatabaseBridge routines, ServiceBridge programs and EasyLinkage programs. If this property is not set, then the same directory as the source file is assumed as root directory for the subfolders.

Property	Meaning

`iscobol.compiler.gui.<control-type>.defaults`	Specifies the default properties and styles for graphical controls.
	
	When the Compiler finds a graphical control in the source, the code specified by this configuration entry is placed at the beginning of the control description in order to be overridden by the original COBOL code if a conflict occurs.
	
	Both SCREEN SECTION and DISPLAY statement are affected.
	
	Valid values for control-type are:
	bar
	bitmap
	check_box
	combo_box
	date_entry
	entry_field
	frame
	grid
	java_bean
	label
	list_box
	push_button
	radio_button
	ribbon
	scroll_bar
	slider
	status_bar
	tab_control
	tree_view
	web_browser
	window
	tool_bar
	
	Example:
	
	Let's consider these two labels:
	
	`03 label`
	`   line 2 col 2 title "label 1".`
	`03 label`
	`   line 4 col 2 title "label 2" font medium-font.`
	
	Setting *iscobol.compiler.gui.label.defaults=color 5 font small-font* makes the Compiler translate the above code as follows:
	
	`03 label`
	`   color 5 font small-font`
	`   line 2 col 2 title "label 1".`
	`03 label`
	`   color 5 font small-font`
	`   line 4 col 2 title "label 2" font medium-font.`

Property	Meaning
`iscobol.compiler.indd`	When this property is set, all format 1 ACCEPT statements which either have no FROM option or specify FROM SYSIN are transformed into READ statements, reading from the file specified by this property. Example: `iscobol.compiler.indd=in.txt` If a relative path is provided, it's resolved according to the runtime working directory. In thin client environment, the file is searched client side. If the file is not found, the system input (SYSIN) is used. The name of the indd file can be changed at runtime via configuration if iscobol.file.env_naming (boolean) is set to true. The management of the file can be performed by a custom file handler specified by the configuration property iscobol.file.indd.
`iscobol.compiler.iss_julian_base`	Specifies the julian base date in the format YYYYMMDD. This property affects only iss dictionaries generated by the -efc option. If this property is not set, no base date is registered in the iss dictionary and c-treeSQL will use March 1st 1700 as base date for julian dates.
`iscobol.compiler.javac`	Specifies the name of the Java compiler to be used in place of the default. This property is ignored by the IDE. When you work with the IDE, the Java compiler has to be configured in the Preferences.
`iscobol.compiler.javac.options`	List of options that should be passed to the Java compiler. Multiple options must be separated by space. A possible values for this property is: `-source 1.8 -target 1.8` The above options create classes that are compatible with previous verions of Java (in this case 1.8 and higher) Options can also be passed to the Java compiler through the -jo compiler option. This property is ignored by the IDE. When you work with the IDE, the Java compiler options have to be configured in the Preferences.
`iscobol.compiler.max_constants *`	This property sets the maximum number of constants that can be generated into a class when the -big option is used. When the number is exceeded, the program is split into multiple class files. The default value is 1000
`iscobol.compiler.max_hostvars *`	This property sets the maximum number of host variables that can be used in an ESQL statement. When the number is exceeded, the program is split into multiple class files. The default value is 700

Property	Meaning
`iscobol.compiler.max_paragraphs *`	This property sets the maximum number of paragraphs that can be generated into a class when the -big option is used. When the number is exceeded, the program is split into multiple class files. The default value is 300
`iscobol.compiler.messagelevel.(error-number)=(action)`	This property allows you to choose which errors should be traced when compiling. The (error-number) is the number that appears at the beginning of the error message. (action) can be one of the following values: **0** = Do not show **1** = Show as Informational **2** = Show as Warning **3** = Show as Error **4** = Show as Severe Error For example, to map the following Error (--E: 154 End statement required END-EVALUATE;) as Informational, this setting must be used: `iscobol.compiler.messagelevel.154=1` Refer to Error numbers list for the list of error messages and their number. **Note -** if error-number refers to a Severe error, then the property will have no effect, as it's not possible to downgrade the severity of a Severe error.
`iscobol.compiler.oop.trim_parameters`	**True** = Alphanumeric data items mapped to java.lang.String parameters in object oriented programming are automatically trimmed. **False** = Alphanumeric data items mapped to java.lang.String parameters in object oriented programming are not automatically trimmed. The default value is True.
`iscobol.compiler.options *`	Lists compiler options. Refer to the Compiler Options section for further details. These options are used along with the ones specified on the command-line. Multiple options must be separated by space. Example: `iscobol.compiler.options=-b -cudc`

Property	Meaning

`iscobol.compiler.outdd`	When this property is set, all format 1 DISPLAY statements which either have no UPON option or specify UPON SYSOUT are transformed into WRITE statements, writing to the file specified by this property. Along with the file name it's possible to specify two optional parameters: the record size and the file type. The file type can be either "L" for Line Sequential (default) or "R" for Record Sequential. Parameters must be separated by space.
	
	Example of setting with just the file name:
	
	`iscobol.compiler.outdd=out.txt`
	
	Example of setting with all parameters:
	
	`iscobol.compiler.outdd=out.txt 10 L`
	
	If a relative path is provided, it's resolved according to the runtime working directory. In thin client environment, the file is searched client side.
	If the file can't be created, the system output (SYSOUT) is used.
	
	The file is initialized when the JVM starts and remains locked until the JVM terminates. If multiple COBOL programs refer to the same outdd file, the information that they display is appended to the existing content in the file.
	
	The name of the outdd file can be changed at runtime via configuration if iscobol.file.env_naming (boolean) is set to true.
	
	The management of the file can be performed by a custom file handler specified by the configuration property iscobol.file.outdd.

`iscobol.compiler.regexp *`	This property allows you to modify the source code on the fly at compile time through regular expressions. See Source code preprocessing for details.
`iscobol.compiler.rundebug`	This property specifies how the Compiler interacts with the remote debuggers. It allows you to debug preoprocessor programs written in COBOL. See Preprocessor programs for more information. Possible values are: **0** = Remote debugging is not possible. **1** = The first preprocessor program will start and run immediately. The remote debug session waits and listens for a program that is compiled for debug to launch, connecting to the program when it detects it. **2** = The Compiler will start in debug mode, pausing to connect to a remote debugger before running the first preprocessor program. The default value is 0.

Properties for the EasyLinkage feature

Property	Meaning
`iscobol.compiler.easylinkage (boolean)`	**0** = The EasyLinkage feature is turned off. No bridge classes are generated. **1** = The EasyLinkage feature is turned on and it generates bridge classes for each COBOL program with Linkage Section. **2** = The EasyLinkage feature is turned on and it generates stub classes for each CALL statement. **3** = The EasyLinkage feature is turned on and it generates both bridge classes and stub classes. The default value is 0.
`iscobol.compiler.easylinkage.cut`	This property specifies the substrings to be removed from the Linkage Section data items name before creating the corresponding Java object in the bridge program. Multiple values must be separated by space. For example: if a data item is named "lnk-p1" and this property is set to "lnk-", then the item will be considered as "p1".
`iscobol.compiler.easylinkage.decoration (boolean)`	**True** = Linkage Section data items name is capitalized according to Java rules in order to build the name of the corresponding Java object in the bridge program. **False** = Linkage Section data items name is normalized according to COBOL rules (it's made uppercase replacing hyphens with underscores) in order to build the name of the corresponding Java object in the bridge program. For example: if a data item is named "lnk-p1", the corresponding Java object will be named "lnkP1" if this property is set to true, "LNK_P1" otherwise. The default value is True.
`iscobol.compiler.easylinkage.package`	This property specifies the package that the generated classes must belong to. By default, no package is used.
`iscobol.compiler.easylinkage.prefix`	This property specifies a prefix to be put before the name of the generated bridge class. For example: if the compiled program is named "PROG1" and this property is set to "bridgeTo", then the generated class will be named "bridgeToPROG1". The default value is "link".

DatabaseBridge (EasyDB) Configuration

Property	Meaning
`iscobol.compiler.easydb (boolean)`	**True =** The Compiler generates bridge classes that allow the COBOL programs to access RDBMS in the same way as they do with ISAM indexed files. **False =** No bridge class is generated. The default value is False.

Property	Meaning
`iscobol.compiler.easydb.db2 (boolean)`	**True** = Generate EDBI routines suitable for IBM DB2. **False** = Don't generate EDBI routines suitable for IBM DB2. The default value is False.
`iscobol.compiler.easydb.db2_as400 (boolean)`	**True** = Generate EDBI routines suitable for IBM DB2 on AS/400. **False** = Don't generate EDBI routines suitable for IBM DB2 on AS/400. The default value is False.
`iscobol.compiler.easydb.db2_as400.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.db2_as400*=true. The default value is "d24".
`iscobol.compiler.easydb.db2.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.db2*=true. The default value is "db2".
`iscobol.compiler.easydb.defchar (boolean)`	**True** = Map alphanumeric COBOL fields to CHAR on the database. **False** = Map alphanumeric COBOL fields to VARCHAR on the database. The default value is False.
`iscobol.compiler.easydb.duplicates_in_order (boolean)`	**True** = Return records with duplicate key values in the primary key order. **False** = Return records with duplicate key values as you read them from the database. The default value is False.
`iscobol.compiler.easydb.entry_points (boolean)`	**True** = Generate entry-points in the EDBI routine where the user can inject customized code. **False** = Don't generate entry-points. See Extending EDBI routines through entry points for more information. The default value is False.
`iscobol.compiler.easydb.generic (boolean)`	**True** = Generate generic EDBI routines. **False** = Don't generate generic EDBI routines. The default value is False, however generic EDBI routines are generated by default if *compiler.easydb* is true and all the *easydb.<rdbms>* properties are false.
`iscobol.compiler.easydb.generic.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.generic*=true. The default value is "gen".

Property	Meaning
`iscobol.compiler.easydb.high_values_as_max_val (boolean)`	**True** = Replace high-values with the maximum value allowed by the field. **False** = Don't replace high-values in the 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. The default value is False.
`iscobol.compiler.easydb.index_only (boolean)`	**True** = Generate EDBI routines only for indexed files. **False** = Generate EDBI routines for indexed, line sequential, relative and sequential files. The default value is True.
`iscobol.compiler.easydb.informix (boolean)`	**True** = Generate EDBI routines suitable for Informix **False** = Don't generate EDBI routines suitable for Informix. The default value is False.
`iscobol.compiler.easydb.informix.dates_as_strings (boolean)`	**True** = Use the string representation to deal with dates. **False** = Use conversion functions to deal with dates. The default value is False. This property is considered only for Informix.
`iscobol.compiler.easydb.informix.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.informix*=true. The default value is "ifx".
`iscobol.compiler.easydb.isam_eof (boolean)`	**True** = Use ISAM positioning rules on end of file. **False** = Don't use ISAM positioning rules on end of file. The default value is False.
`iscobol.compiler.easydb.julian_routines=<cbdb>;<dbcb>`	This property specifies the name of custom routines for julian dates management. Two routines must be provided: the first is for conversions between COBOL and database while the second is for conversions between database and COBOL. The two names must be separated by semicolon. In order to create these routines, you can edit and customize EDBI-DTJUCBDB.cbl and EDBI-DTJUDBCB.cbl installed under $ISCOBOL/easydb/edbisource. If the property is not set, then the standard EDBI routines EDBI-DTJUCBDB and EDBI-DTJUDBCB stored in the isCOBOL runtime library are used.

Property	Meaning
`iscobol.compiler.easydb.light_cursors`	This property is considered only for MySQL and PostgreSQL. When set to "1" or "2", the EDBI routines perform a pagination of the read records. Possible values are: **0** = No pagination **1** = Pagination only when using a UNIQUE index **2** = Pagination for every index The default value is 0. **Note -** An additional column named OID is generated in the tables when the value 2 is used. For this reason, routines generated with *light_cursors=2* can't work on tables that were created by routines generated with a different value of *light_cursors* and vice versa. The number of records per page is controlled by the configuration properties iscobol.easydb.mysql_row_limit and iscobol.easydb.postgres_row_limit respectively.
`iscobol.compiler.easydb.max_char_len`	This property sets the size limit used by EDBI routines to choose if the an alphanumeric COBOL field must be mapped to CHAR or VARCHAR. If the size of the COBOL field is not greater than the value of the property, then the field is mapped to CHAR, otherwise it is mapped to VARCHAR. The default value is 0.
`iscobol.compiler.easydb.mysql (boolean)`	**True** = Generate EDBI routines suitable for MySQL. **False** = Don't generate EDBI routines suitable for MySQL. The default value is False.
`iscobol.compiler.easydb.mysql.hints (boolean)`	This property is considered only for MySQL. It forces the use of hints within queries used to simulate the COBOL Start statement. This should speed up the reading of records. Possible values are: **True** = Use hints **False** = Don't use hints The default value is False.
`iscobol.compiler.easydb.mysql.oid_name`	This property specifies the name of the OID column generated when iscobol.compiler.easydb.light_cursors is set to 2. It's considered only for MySQL. The default value is "OID".
`iscobol.compiler.easydb.mysql.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.mysql*=true. The default value is "mys".

Property	Meaning

iscobol.compiler.easydb.names_with_leading_zeros (boolean)	**True =** 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.
	False = Don't use leading zeroes in OCCURS item names.
	
	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.`
	
	If the property is false, then the columns 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`
	
	If the property is true, then 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`
	
	The default value is False.

iscobol.compiler.easydb.no_check (boolean)	**True =** Don't check for table existence during OPEN. **False =** Check for table existence during OPEN. The default value is False. **Note -** disabling the check of table existence may improve performance, especially if your programs use the OPEN statement a lot. However, there are also few side effects, for example, since the OPEN will never fail with 'file not found' you will not be able to create with I$IO having io_creates=1 in the configuration. Also, if the table doesn't exist but the OPEN doesn't fail, you might have odd errors in the next operations.
iscobol.compiler.easydb.oracle (boolean)	**True =** Generate EDBI routines suitable for Oracle. **False =** Don't generate EDBI routines suitable for Oracle. The default value is False.
iscobol.compiler.easydb.oracle.hints	This property is considered only for Oracle. It forces the use of hints within queries used to simulate the COBOL Start statement. This should speed up the reading of records. Possible values are: **0 =** Don't use hints. **1 =** Use hints but keep the ORDER BY in they query **2 =** Use hints and discard the ORDER BY in the query, assuming that records will be sorted according to the index set by the hint. The default value is 0.

Property	Meaning
`iscobol.compiler.easydb.oracle.index_storage_initial_value`	This property is considered only for Oracle. It specifies the initial storage value for indexes. If not set, then the default storage value is used.
`iscobol.compiler.easydb.oracle.index_storage_next_value`	This property is considered only for Oracle. It specifies the next storage value for indexes. If not set, then the default storage value is used.
`iscobol.compiler.easydb.oracle.index_storage_pctincrease_value`	This property is considered only for Oracle. It specifies the initial pctincrease for indexes. If not set, then the default pctincrease value is used.
`iscobol.compiler.easydb.oracle.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.oracle*=true. The default value is "ora".
`iscobol.compiler.easydb.oracle.table_storage_initial_value`	This property is considered only for Oracle. It specifies the initial storage value for tables. If not set, then the default storage value is used.
`iscobol.compiler.easydb.oracle.table_storage_next_value`	This property is considered only for Oracle. It specifies the next storage value for tables. If not set, then the default storage value is used.
`iscobol.compiler.easydb.oracle.table_storage_pctincrease_value`	This property is considered only for Oracle. It specifies the initial pctincrease for tables. If not set, then the default pctincrease value is used.
`iscobol.compiler.easydb.oracle.tablespace_index_name`	This property is considered only for Oracle. It specifies the name of the tablespace where indexes must be created and searched. If not set, then the default tablespace is used.
`iscobol.compiler.easydb.oracle.tablespace_name`	This property is considered only for Oracle. It specifies the name of the tablespace where tables must be created and searched. If not set, then the default tablespace is used.
`iscobol.compiler.easydb.oracle.wait_for_lock`	This property is considered only for Oracle. It specifies how the EDBI routine should behave when a record locked condition occurs. You can choose between waiting for the lock to be released or returning an error to the COBOL program. Possible values are: **0** = Never wait for locks and return an error instead **1** = Always wait for locks **2** = Wait for locks or return an error depending on the iscobol.easydb.wait_for_lock (boolean) runtime setting The default value is 0.

Property	Meaning
`iscobol.compiler.easydb.output`	This property specifies the output directory where the EDBI routine must be generated. If not set, then the routine is generated in the same directory as the COBOL source file.
`iscobol.compiler.easydb.postgres (boolean)`	**True** = Generate EDBI routines suitable for PostgreSQL. **False** = Don't generate EDBI routines suitable for PostgreSQL. The default value is False.
`iscobol.compiler.easydb.postgres.oid_name`	This property specifies the name of the OID column generated when iscobol.compiler.easydb.light_cursors is set to 2. It's considered only for PostgreSQL. The default value is "OID".
`iscobol.compiler.easydb.postgres.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.postgres*=true. The default value is "pgs".
`iscobol.compiler.easydb.sql (boolean)`	**True** = Generate a script file with .sql extension that includes the CREATE TABLE statement. **False** = Don't generate script files. The default value is False.
`iscobol.compiler.easydb.sql.output`	This property specifies the output directory where the script files must be generated. If not set, then the script files are generated in the same directory as the COBOL source file. This property is considered only if iscobol.compiler.easydb.sql (boolean) is set to true.
`iscobol.compiler.easydb.sqlserver (boolean)`	**True** = Generate EDBI routines suitable for Microsoft SQL Server. **False** = Don't generate EDBI routines suitable for Microsoft SQL Server. The default value is False.
`iscobol.compiler.easydb.sqlserver.datetime_always (boolean)`	**True** = Every field with EFD DATE directive becomes a DATETIME, regardless of the date format string. **False** = Fields with EFD DATE directive become DATE, TIME or DATETIME according to the date format string. The default value is False. This property is considered only for Microsoft SQL Server.
`iscobol.compiler.easydb.sqlserver.latin1_general_bin`	This property is considered only for Microsoft SQL Server. It forces the use of latin1_general_bin collating sequence, ensuring that data is ordered according to the ASCII value of the characters. Possible values are: **0** = Don't use the latin1_general_bin collating sequence **1** = Use latin1_general_bin during CREATE TABLE **2** = Use latin1_general_bin during ORDER BY The default value is 0.

Property	Meaning
`iscobol.compiler.easydb.sqlserver.prefix`	This property specifies the prefix that will be put before each indexed file name to build the bridge program name when *easydb.sqlserver*=true. The default value is "srv".
`iscobol.compiler.easydb.test_not_numeric`	This property allows you to write zeros instead of non-numeric values in numeric key fields, avoiding data conversion errors that might occur with some JDBC drivers. Possible values are: **0** = No action **1** = Write zero instead of non-numeric values **2** = Same as "1", but with trace. The trace is stored in the file pointed by the iscobol.edbi.notnum.tracefile runtime setting. The default value is 0.
`iscobol.compiler.easydb.unlock_all (boolean)`	**True** = Enable the support for UNLOCK ALL statement. **False** = It's not possible to perform UNLOCK ALL. The default value is False.

Property	Meaning
`iscobol.compiler.servicebridge (boolean)`	**True** = The Compiler generates bridge classes that allow the COBOL programs to be used as a Web Service. **False** = No bridge class is generated. The default value is False.
`iscobol.compiler.servicebridge.bean`	This property specifies the service type for which a bean client program should be generated along with the bridge program. Valid values are: **SOAP** = Generate beans for SOAP Web Services **REST** = Generate beans for REST Web Services
`iscobol.compiler.servicebridge.bean.package`	This property specifies the package for the bean class. By default, there's no package.
`iscobol.compiler.servicebridge.bean.prefix`	This property specifies the prefix to be applied to the bean client program name. The default value is "bean".
`iscobol.compiler.servicebridge.bean.url`	This property specifies the URL to which the bean client program must connect. The default value is "http://localhost:8080/services"
`iscobol.compiler.servicebridge.type`	This property specifies what type of service will be provided by the bridge classes. Possible values are: **SOAP** = SOAP Web Service **REST** = REST Web Service The default value is "SOAP".
`iscobol.compiler.servicebridge.rest.prefix`	This property specifies the prefix that will be put before the COBOL program name to build the bridge program name when servicebridge.type=REST. The default value is "rest".
`iscobol.compiler.servicebridge.soap.charset`	This property specifies the encoding that should be used int the SOAP service response. All the canonical names listed in the following Java documentation can be used as value for this property: https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html The default value is "UTF-8"
`iscobol.compiler.servicebridge.soap.namespace_suffix`	This property specifies the string to be appended to the value of iscobol.compiler.servicebridge.soap.namespace_suffix in order to compose the final namespace. By default, the COBOL program name is used.

Property	Meaning
`iscobol.compiler.servicebridge.soap.prefix`	This property specifies the prefix that will be put before the COBOL program name to build the bridge program name when iscobol.compiler.servicebridge.type=SOAP. The default value is "soap".
`iscobol.compiler.servicebridge.soap.url`	This property specifies the Web Service base URL. The default value is "http://localhost:8080"
`iscobol.compiler.servicebridge.soap.style`	This property specifies the SOAP messaging style. Possible values are: **Document** = use Document style **RPC** = use RPC style The Document style indicates that the SOAP body contains a XML document which can be validated against pre-defined XML schema document. RPC indicates that the SOAP message body contains an XML representation of a method call and uses the names of the method and its parameters to generate XML structures that represent a method's call stack. The default value is "RPC".
`iscobol.compiler.servicebridge.soap.namespace`	This property specifies the unique namespace in order for client applications to distinguish your SOAP Web Service from other services on the Web. The default value is "http://tempuri.org"
`iscobol.compiler.servicebridge.package`	This property specifies a package for the generated bridge class. By default, the bridge classes have no package.

General Runtime Configuration

(*) The asterisk after the property name means that the property is static, it's inquired only once by the Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.array_cache`	This property represents the number of OCCURS elements stored in memory for increasing performances. Setting this property to 0 or to a negative value, disables the array caching feature. The default value is 101.

Property	Meaning
`iscobol.array_check *`	**-1** = Array boundaries are checked at Runtime in order to provide more details in case of "out of bounds" errors. If a program addresses an item that is outside the valid range, an error message is written to the log (iscobol.tracelevel must be set to a value greater than zero). The error message informs about the data item name and the problematic index value. **0** = Array boundaries are not checked. If a program addresses an item that is outside the valid range, a generic "out of bounds" error message is shown and the program exits. The -m1 option may avoid the crash and make the program access the area of the next Working-Storage item instead. **1** = Array boundaries are checked at Runtime in order to provide more details in case of "out of bounds" errors. If a program addresses an item that is outside the valid range, an error message is shown and the program exits. The error message informs about the data item name and the problematic index value. **Note -** for an accurate error message, program shouldn't be compiled with the -ostrip option. The default value is 0.
`iscobol.call_cancel.hook`	This property specifies the name of a class that the runtime will invoke before and after each CALL and CANCEL statement. The class must implement the com.iscobol.rts.CallHandler interface. Refer to the javadoc installed with isCOBOL for more details.
`iscobol.call_run.sync (boolean)`	**True** = The calling program in a CALL RUN waits for the called program to terminate before proceeding. **False** = The calling program in a CALL RUN doesn't wait for the called program to terminate before proceeding. The default value is False.
`iscobol.check.numeric_content *`	**-1** = an exception is written to the log for USAGE DISPLAY numeric and numeric-edited variables that don't contain a number. iscobol.tracelevel must be set to a value greater than zero. **0** = no exceptions are thrown for USAGE DISPLAY numeric and numeric-edited variables that don't contain a number. **1** = an exception is thrown for USAGE DISPLAY numeric and numeric-edited variables that don't contain a number. The default value is 0. The check is performed when the Framework reads bytes from the memory and transform them to a number, for example when the program displays numeric data items or when arithmetic operations are performed. For numeric edited items the exception is thrown if there isn't a numeric digit in correspondence with the '9' characters of the picture.

Property	Meaning
`iscobol.checkdiv *`	This property allows you to specify an alternate runtime response to a divide by zero condition when the statement does not include a ON SIZE ERROR clause. Valid values are: **-1** = The error message "Attempt to divide by zero" is written to the log (**iscobol.tracelevel** must be set to a value greater than zero) and the program aborts. **0** = Results are undefined. **1** = The program aborts with the error message "Attempt to divide by zero". **2** = The result is zero. **3** = The result is the dividend, as if the division was by 1 instead of by zero. **Note -** for an accurate error message, program shouldn't be compiled with the -ostrip option. The default value is 0.

Property	Meaning
`iscobol.code_prefix`	This property lists the paths and the jar libraries in which programs are searched. Multiple values must be separated by the line feed character or by the current operating system path separator. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". An asterisk can be specified at the end of a path to include all the jars in that path. code_prefix is used by: • CALL STATEMENT • *isrun* and *iscrun* commands • isCOBOL Server • WebDirect CLASSPATH is read first, then code_prefix is read. This means that if the same program is found in both CLASSPATH and code_prefix, it's loaded from CLASSPATH. If code_prefix is not set, CLASSPATH is used instead. Note that Wrappers automatically add the current directory, the jar libraries of the isCOBOL *lib* directory and the jar libraries of the isCOBOL *jars* directory to the Classpath. All classes loaded from code_prefix are loaded into memory each time they are called, if the program cancels these classes from memory (see CANCEL STATEMENT). A class is reloaded from disk only if the disk file last modification timestamp is different from the last modification timestamp of the class loaded in memory. Programs loaded from CLASSPATH, instead, are always stored in memory until the Runtime Framework terminates and the CANCEL STATEMENT just initializes their DATA DIVISION. When code_prefix is set, it's possible to specify a path in the name of the programs, e.g. • CALL "/path/to/PROG1" • isrun path\to\PROG1 COBOL classes (programs that have CLASS-ID in their IDENTIFICATION DIVISION) are loaded from the code_prefix only if they're in the same location as the COBOL program (a program that has PROGRAM-ID in its IDENTIFICATION DIVISION) that invokes them and they're cancelled and reloaded along with that COBOL program. However, loading COBOL classes from the code_prefix may create issues; for example it's not possible to implement the same interface in two different COBOL classes without causing a ClassCastException. Loading COBOL classes from the code_prefix is discouraged. COBOL classes should always be loaded from the CLASSPATH.

Property	Meaning
`iscobol.code_prefix.reload *`	This property can be set along with iscobol.code_prefix to alter the class loading logic. Possible values are: **1** = The runtime accesses the disc in order to find if the program class file changed. If the program class file has changed, the runtime reloads the class from disc. **0** = The runtime reloads from disc only those programs that were unloaded by C$UNLOAD. No check is performed on the program class file last modification date. A dedicated ClassLoader for each program is used, hence you can unload specific programs via the C$UNLOAD routine. CLASS-ID programs are unloaded only along with the program that instantiated them. **2** = The runtime reloads from disc only those programs that were unloaded by C$UNLOAD. No check is performed on the program class file last modification date. A common ClassLoader is used, hence the C$UNLOAD routine always unloads all the programs. This value is particularly useful in a thin client environment to have the next client sessions start fresh with updated classes. In a thin client environment the unload of programs can be managed with the administration panel. See Unloading programs for more information. The default value is 1.
`iscobol.conf *`	This property specifies a configuration file to be used by the Framework. This configuration file is used along with other configuration sources as explained in Configuration. This property can be specified only on the command line. Setting it in a configuration file has no effect.

`iscobol.conf.copy`	This property specifies a configuration file to be added to the current configuration. Relative pathnames are resolved according to the Framework's working directory.
	
	This property should be set in a configuration file.
	Multiple entries of this property can appear in the configuration file.
	The line where *iscobol.conf.copy* is written is the position where the new properties will be included.
	
	This property can't be set dynamically by COBOL programs. If you need to add a configuration file to the current configuration in the COBOL program, use the C$CONFIG routine.
	This property can't be inquired.
	
	Example:
	Suppose having a file named *app.properties* with the following entries:
	
	`iscobol.code_prefix=/opt/myapp/class`
	`iscobol.conf.copy=default.properties`
	`iscobol.file.prefix=/opt/myapp/data`
	
	A runtime session launched with the *app.properties* configuration file will use all the configuration entries in the *default.properties* file plus the *file.prefix* setting and the *code_prefix* setting specified in the *app.properties* configuration file. If *default.properties* includes a *code_prefix* setting, it will override the *code_prefix* setting in *app.properties*. If *default.properties* includes a *file.prefix* setting, it will be overridden by the *file.prefix* setting in *app.properties*.

Property	Meaning
`iscobol.conf.only *`	This property specifies a configuration file to be used exclusively by the Framework. No other configuration sources are inquired when this property is set.

`iscobol.conf.var_delimiters`	This property allows you to set delimiters for free text (comments) and Java variables within configuration property values.
	
	For example, having
	
	`iscobol.conf.var_delimiters=/*.*/`
	`iscobol.jver=/*java.version*/`
	`iscobol.hello=Hello,/* this is a comment */ World!`
	
	the configuration variable *jver* will have the value "1.8.0_261" (assuming that you're working with this JVM) and the configuration variable *hello* will have the value "Hello, World!"
	
	Refer to the table in C$GETENV documentation for a list of Java properties that you can include between delimiters.

`iscobol.crypt.algorithm`	This property specifies the symmetric-key algorithm used by C$DECRYPT and C$ENCRYPT library routines. See the Cipher section in the Java Cryptography Architecture Standard Algorithm Name Documentation for information about standard algorithm names. The default value is "AES".
`iscobol.current_date`	This property changes the format used in CURRENT-DATE (IBM DOS/VS COBOL compatibility). **0** = MM/DD/YY is used. **1** = DD/MM/YY is used. The default value is 0.
`iscobol.default_options`	This property specifies the default command line options used by the runtime (iscrun) and the client (iscclient). When used with iscrun, if the value of this property includes either the -c option or the -conly option and the configuration file pointed to by this option includes another iscobol.default_options setting, such setting is ignored. When used with iscclient, if the value of this property includes the -lc option and the configuration file pointed to by this option includes another iscobol.default_options setting, this second setting of iscobol.default_options is ignored.
`iscobol.default_program`	This property specifies the name of the program to be executed when no program is specified on the command line. The name of the program specified by this property can be followed by a list of parameters separated by space that the program will receive as chaining parameters. Example: `iscobol.default_program=MAIN_MENU company1 user1` In thin client environment the property can be specified either in the remote configuration file or in the local configuration file. If the property is specified in both configuration files, then the value in the local configuration file is considered.

Property	Meaning
`iscobol.dll_convention`	This property sets the convention used when the DLL function is called. **0** = C convention. **1** = PASCAL (or WINAPI) convention. The default value is 0.

`iscobol.encoding *`	This property specifies the encoding used to decode the content of alphanumeric data items. It affects only in-memory conversions.
	Due to this limited effect, it's recommended to use the *file.encoding* Java property instead, as the Java property specifies the encoding for every operation, including i-o.
	
	The valid values for *iscobol.encoding* and *file.encoding* are the canonical names listed at https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html.
	
	iscobol.encoding can appear only in the configuration file. Setting it with the SET ENVIRONMENT statement has no effect.
	
	file.encoding can't appear in the configuration file, because it's not an isCOBOL configuration entry (it lacks the "iscobol." prefix) and can't be set with the SET ENVIRONMENT statement. It must be set on the command line, for example:
	
	`iscrun -J-Dfile.encoding=UTF-8 PROG`
	
	In thin client environment it's good practice to use the same encoding on both client side and server side, for example:
	
	`iscserver -J-Dfile.encoding=CP1252 -hostname 192.168.0.101 -port 10999`
	`iscclient -J-Dfile.encoding=CP1252 -hostname 192.168.0.101 -port 10999 PROG`
	
	The default encoding depends on the operating system.
	
	Note - the *file.encoding* Java property is used also by the Compiler. Use it to compile source files that were saved with a particular encoding. For example, in order to compile a source file saved in UTF-8 format, use:
	
	`iscc -J-Dfile.encoding=UTF-8 PROG.cbl`

`iscobol.floating_point_format *`	This property specifies the format used to store Float and Double values. Possible values are: • ibm_hfp • ieee_754 Under -cm option and -cv option this property affects also COMP-1 and COMP-2 data items, as they're treated as Float and Double respectively. The default value is "ieee_754".

Property	Meaning
`iscobol.display_message`	This property defines how messages are shown to the user. It affects both messages displayed by the program through DISPLAY MESSAGE BOX statement and error messages shown by the runtime system. In thin client environment it is also evaluated on the client side for the Client connection error messages. Possible values are: **0** = All messages are shown in a message box. **1** = All messages are sent to sysout. In a thin client environment, runtime error messages are sent to the server sysout while messages displayed by the program are sent to the client sysout. **2** = All messages are sent to syserr. In a thin client environment, runtime error messages are sent to the server syserr while messages displayed by the program are sent to the client syserr. **3** = Messages displayed by the program are sent to the syserr (client syserr in thin client) while runtime error messages are printed to a file named <program_name><number>.ads.log (where <number> is a progressive number calculated by the runtime). The file name and location can be customized via the iscobol.exception.dumpfile property. Any other value is equivalent to 2. The default value is 0. (iscobol.exception_message is still supported for backward compatibility, but this property affects only messages produced after the first COBOL program has been loaded)
`iscobol.display_message_timeout`	This property specifies a timeout for the error message boxes generated by the runtime system. The timeout is expressed in hundreds of second. When the timeout expires, the error message box is automatically closed as if the user pressed the OK button. In thin client environment, setting the property in the local configuration file of the isCOBOL Client also affects the error messages generated by the isCOBOL Client, such as "Connection timed out" or "Session terminated by the server". By default there is no timeout and error message boxes stay on video until the user closes them.
`iscobol.exception.dump (boolean)`	**True** = Produces "Abend Diagnostic Snapshot" (ADS) in addition to Java exceptions **False** = No ADS is produced in addition to Java exceptions The resulting exception is shown on video or printed to file according to the properties iscobol.display_message and iscobol.exception.message. The default value is False

Property	Meaning
`iscobol.exception.dumpfile`	This property specifies the pathname of the file generated by setting iscobol.display_message=3 or iscobol.exception.message=3. The following special characters are supported in the value of this property: %p, program name %d, current date in the form YYYYMMDD %t, current time in the form HHMMSSTTT %u, username %h, hostname If the value begins with the "+" character, then the report is appended to the specified file, otherwise the new report overwrites the specified file. Example: iscobol.exception.dumpfile=/tmp/%p.dump If this property is not set, a file named <program_name><number>.ads.log (where <number> is a progressive number calculated by the runtime) is generated by default in the working directory. (iscobol.exception.prefix is still supported for backward compatibility)
`iscobol.exception.java (boolean)`	**True** = Internal java methods are traced in exception messages. **False** = COBOL paragraph names are traced in exception messages. The default value is False.
`iscobol.exception.message`	This property defines how exception messages are shown to the user. It affects only error messages shown by the runtime system. **0** = Messages are shown in a message box. **1** = Messages are sent to sysout. In a thin client environment, they're sent to the server sysout. **2** = Messages are sent to syserr. In a thin client environment, they're sent to the server syserr. **3** = Messages are printed to a file named <program_name><number>.ads.log (where <number> is a progressive number calculated by the runtime). The file name and location can be customized via the iscobol.exception.dumpfile property. Any other value is equivalent to 2. The default value is 0. Note that only the messages produced after the first COBOL program has been loaded (e.g. a ArrayOutOfBoundsException generated by OCCURS overflow) are affected by this setting. In order to affect all messages (e.g. a invalid command line error) set iscobol.display_message instead.
`iscobol.help_program`	This property identifies the COBOL program to be called for item's help. The program is called with a implicit CALL statement so the program class can be loaded either from the Class Path or from the iscobol.code_prefix. See Help automation for more details.

Property	Meaning
`iscobol.help_program_mouse_stop_delay`	This property enables the invocation of the help program specified by iscobol.help_program when the user leaves the mouse pointer over a control. The value of this property is the number of milliseconds that the runtime must wait before invoking the help program. Setting this property to 0 disables the feature. In this case the help program can be invoked only via keyboard. See Help automation for more details. The default value is 0.

`iscobol.hot_key.`*`ProgramName`*	This property associates an exception value or a range of exception values with a hotkey program. The hotkey program will automatically be executed when the exception occurs. To define a range of values specify the minimum value followed by a dash and the maximum value.
	
	Example:
	
	`iscobol.hot_key.myprog=100`
	
	(myprog will run automatically when the crt status is 100).
	
	`iscobol.hot_key.myprog=1-5`
	
	(myprog will run automatically when crt-status is 1, 2, 3, 4 or 5).

`iscobol.jvm_options`	This property specifies the command-line options to be passed to a new Java Virtual Machine (JVM) launched by a running isCOBOL process. isCOBOL instantiates new JVMs in these two cases: • when the -d option is used to run in debug. The Debugger starts a new JVM to run the program. • when iscobol.as.multitasking is set to a value greater than 0, causing the Application Server to start a separate JVM process for each connection. Multiple options must be separated by a space. For example, you can change the default value of -Xmx128m and -Xmx128m with this property setting: `iscobol.jvm_options=-Xms1024m -Xmx124m` **Note -** This property does not affect currently running processes because the configuration variables are read after the initial JVM is already started. The command-line is the only place you can set options for the initial JVM. The default value is "-Xms128m -Xmx128m".
`iscobol.literal.numeric.comp (boolean)`	**True =** literals are treated as USAGE COMP in LENGTH OF functions and other integer functions. **False =** literals are treated as USAGE DISPLAY in LENGTH OF functions and other integer functions. The default value is False.

Property	Meaning
`iscobol.little_endian (boolean)*`	**True** = Native numeric data items are stored in Little Endian format. **False** = Native numeric data items are stored in Big Endian format. The default value is system dependent. This property is deprecated and shouldn't be set as the current JVMs automatically use the correct endianness.
`iscobol.logclass`	This property specifies an alterante class to manage the trace of the runtime activity. The class must implement the *com.iscobol.logger.Logger* interface. See Slf4jLogger class (com.iscobol.logger.Slf4jLogger) for further details.
`iscobol.logfile`	This property specifies the path of the log file. Backslashes must be doubled. For example, the path "C:\MyLogDir\MyLogFile" would be "C:\\MyLogDir\\MyLogFile". Note: To produce a trace log, set iscobol.tracelevel to a non-zero value. The isCOBOL framework uses the java.util.logging package, and there are many configuration options. For example, you can specify "%h" in the iscobol.logfile and it will be replaced by the user's home directory. You can specify %yyyy, %mm, %dd, %hh, %nn, %ss and %cc and they will be replaced with the current year, month, day, hour, minute, second and hundredth of second respectively. You can specify a "%u" in the iscobol.logfile and it will be replaced with a unique number at runtime to resolve conflicts. The %u is replaced by a unique number, 0, 1, 2, ... The logic to determine the unique number is to use the lowest number that is not in current use by a process. The log files are "locked" by creating a ".lck" file, and are unlocked by deleting that ".lck" file. So if the filename is fred%u.log and fred0.log.lck exists, the process will create fred1.log (and fred1.log.lck). If fred0.log.lck does not exist then the process will overwrite fred0.log. It does not get appended to. (The javadoc for FileHandler says "If the FileHandler tries to open the filename and finds the file is currently in use by another process it will increment the unique number field and try again. This will be repeated until FileHandler finds a file name that is not currently in use") See https://docs.oracle.com/javase/8/docs/api/index.html?java/util/logging/FileHandler.html for other pattern components and logging properties. To include a process ID in the log filename on UNIX/Linux, create a shell script and use $$ to substitute the process id of the current shell. For example, to create a log file named myapp followed by an underscore and the process id of the shell, specify "-Discobol.logfile=myapp_$$.log" on your java command line.

Property	Meaning

	Note that these log files will accumulate until they are deleted or until the process id wraps around.
	
	If you do not set iscobol.logfile then the trace log will be written to $ISCOBOL/bin/isrun.log where $ISCOBOL is the isCOBOL installation directory.
	
	The iscobol.logfile value should not be enclosed in double-quotes, even if there are embedded spaces in the path. On Windows, you can use forward slashes or double-backslashes. For example, any of the following will work:
	
	`iscobol.logfile=C:\\parent dir\\sub dir\\myapp.log`
	`iscobol.logfile=C:/parent dir/sub dir/myapp.log`
	`iscobol.logfile=/parent dir/sub dir/myapp.log`
	`iscobol.logfile=%h/myapp%u.log`
	
	To include portions of the current date and time in the log path name, you can rely on the following patterns:
	
	"%yyyy" the current year
	"%mm" the current month (01-12)
	"%dd" the current day (01-31)
	"%hh" the current hour (00-23)
	"%nn" the current minute (00-59)
	"%ss" the current second (00-59)
	"%cc" the current hundred of second (00-99)
	
	If these patterns are used in the directory part of the path name, the runtime takes care of creating the necessary subfolders, if they don't exist.
	
	For example, if you set "iscobol.logfile=%h/logs/%hh%nn/iscobol.log" and launch the runtime at 3:01 PM, you will obtain the following log file: /home/username/logs/1501/iscobol.log.
	
	In multithread environments, a separate log file for each thread is generated. The name of these log files has the format <name>.#, where <name> is the value of the *iscobol.logfile* property and # is a ordinal number assigned by isCOBOL to each new thread.

`iscobol.logfile.append`	**True =** Append content to the existing log file, if it exists **False =** Overwrite the existing log file, if it exists The default value is False.
`iscobol.logfile.maxlen`	This property specifies the maximum number of bytes that can be written in the log file. When this number is reached, the runtime cleans the log file before writing new information. You can set iscobol.logfile.number to a value greater than 1 in order to preserve one of more copies of the log files before they're erased. By default there's no limit in the size of the log file.

Property	Meaning
`iscobol.logfile.number`	This property specifies how many log files can be kept on disk when iscobol.logfile.maxlen is set. The name of these log files is a concatenation between the value of iscobol.logfile and a progressive number starting from zero. It's good practice to set this property to a value not less than 2 in order to preserve a bit of history of the program activity. The default value is 1, that means only one log on disk.
`iscobol.memory.alpha_edited (boolean)`	**True =** Accept and VALUE clause on alphabetic-edited and alphanumeric-edited items overwrites editing characters. **False =** Editing characters in alphabetic-edited and alphanumeric-edited items are always preserved. For example, having the following data item: `77 VAR1X-E PIC X/X/X VALUE "ABC".` If the property is true, VAR1X-E will contain "ABC" otherwise it will contain "A/B/C". The program accepts VAR1X-E and the user types "AAA". If the property is true, VAR1X-E will contain "AAA" otherwise it will contain "A/A/A". The default value is False.
`iscobol.os.name`	This property specifies the value that is returned in the corresponding data item of the SYSTEM-INFORMATION group item.
`iscobol.properties.acu_compat (boolean)`	**True =** Activates Acucobol-GT compatibility on configuration, that means: • environment variables have precedence on configuration properties • file name translation is repeated until no new translation is found • OPENSAVE-BROWSE-FOLDER starts from Computer rather than the current directory if opnsav-default-dir isn't set **False =** Doesn't activate Acucobol-GT compatibility on configuration The default value is False.
`iscobol.recursion_data_global (boolean) *`	**True =** Working-Storage and FD data are shared between programs called in recursion. **False =** Each recursive program has its own Working-Storage and FD. The default value is True.

Property	Meaning

`iscobol.remote.code_prefix`	This property calls remote programs by specifying the hostname and port on which an isCOBOL Server is listening. Values must begin with "isc://". Multiple values must be separated by "\n". For example:
	
	`isc://hostname1:portnumber1\nisc://hostname2:portnumber2`
	
	synchronous CALLs are executed by default.
	
	asynchronous CALLs are also supported using the syntax CALL THREAD.
	
	CALLs to client programs using the syntax CALL CLIENT as well as opening data files client-side by assigning them to the class "com.iscobol.io.RemoteRelative" are not supported. Remote programs cannot access client resources as it happens in the standard thin client environment.
	
	If the called programs were compiled with -cp, then the iscp protocol must be used instead of isc, e.g.
	
	`iscp://hostname1:portnumber1\nisc://hostname2:portnumber2`

`iscobol.resource.file`
`iscobol.resource.country`
`iscobol.resource.language`
`iscobol.resource.variant`
	These properties are used to define the name of the resource file for localization.
	
	The name is composed as follows (square brackets enclose optional elements):
	file[_language[_country[variant]]].properties
	
	These properties are loaded along with the program so if you set them dynamically with SET ENVIRONMENT they will affect the called programs and not the current program.
	A sample of this feature is installed with isCOBOL in the folder $ISCOBOL_HOME/sample/multilanguage.
	
	These properties affect also the language of message box buttons and framework dialogs (as described at Language in message boxes and framework dialogs) and the default print page format (as described at winprint-curr-papersize).

`iscobol.rm.development_mode (boolean)`	**True =** Activates the RM/COBOL development mode. **False =** Disables the RM/COBOL development mode. The default value is False. **Note -** this setting has currently no effect.
`iscobol.rundebug *`	This property specifies how the Runtime interacts with the remote debuggers. **0** = Remote debugging is not possible. **1** = The first program will start and run immediately. The remote debug session waits and listens for a program that is compiled for debug to launch, connecting to the program when it detects it. This is useful when you only want to debug some of your application's programs. **2** = The runtime framework will start in debug mode, pausing to connect to a remote debugger before running the program. This is useful when all your programs are compiled for debug and you want to start your remote debugger at the very first line of the application. The default value is 0.

Property	Meaning
iscobol.runtime.compile_flags.mandatory *	List of compile options that must have been used. Programs not compiled with these options will not be executed.
iscobol.runtime.compile_flags.prohibited *	List of compile options that couldn't be used. Programs compiled with these options will not be executed.
iscobol.runtime.currency *	This property specifies the currency sign. Set it to the desired character. The character specified by this property will be used as currency sign regardless of the CURRENCY phrase in the Special-Names. The program must be compiled with the -sdcs option for this feature to take effect.
iscobol.runtime.decimal_point_is_comma (boolean) *	**True** = Every program compiled with the -sddp option shows the comma as the decimal separator and the dot as the thousand separator, regardless of the DECIMAL-POINT clause in the Special Names. **False** = Every program compiled with the -sddp option shows the dot as the decimal separator and the comma as the thousand separator, regardless of the DECIMAL-POINT clause in the Special Names. The default value is False.
iscobol.runtime.native.dynamic.ignore_errors (boolean) *	**True** = No errors are returned if the dyncall native library cannot be loaded during startup. **False** = An error is returned if the dyncall native library cannot be loaded during startup. The default value is True. Since the cause of the error may be shown at the bottom of the exception stack, when you set this property to false it's suggested you also set iscobol.exception.java (boolean) to true. (iscobol.runtime.native.ignore_errors is supported for backward compatibility)
iscobol.runtime.native.static.ignore_errors (boolean) *	**True** = No errors are returned if the stacall native library cannot be loaded during startup. **False** = An error is returned if the stacall native library cannot be loaded during startup. Since the cause of the error may be shown at the bottom of the exception stack, when you set this property to false it's suggested you also set iscobol.exception.java (boolean) to true. The default value is True.
iscobol.runtime.version	This property returns the version number of the Runtime Framework.
iscobol.shared_dlopen_null (boolean)	**True** = C functions called by the COBOL program are also searched for in the current process. **False** = C functions called by the COBOL program are not searched for in the current process. The default value is True.

Property	Meaning
`iscobol.shared_library_list *`	List of dynamic libraries that should automatically be loaded at startup. This feature allows you to avoid calling the library name before using its functions in the program. Multiple paths must be separated by \n character sequence or by the current operating system path separator.
`iscobol.station`	This property specifies the value that is returned in the corresponding data item of the SYSTEM-INFORMATION group item.
`iscobol.substring.check *`	**-1 =** String boundaries are checked at Runtime in order to provide more details in case of "out of bounds" errors. If a program addresses an invalid character position, an error message is written to the log (iscobol.tracelevel must be set to a value greater than zero). The error message informs about the data item name and the problematic character position. **0 =** String boundaries are not checked. If a program addresses an item that is outside the valid range, a generic "out of bounds" error message is shown and the program exits. The -m1 option may avoid the crash and make the program access the area of the next Working-Storage item instead. **1 =** String boundaries are checked at Runtime in order to provide more details in case of "out of bounds" errors. If a program addresses an invalid character position, an error message is shown and the program exits. The error message informs about the data item name and the problematic character position. **Note -** for an accurate error message, program shouldn't be compiled with the -ostrip option. The default value is 0. (iscobol.substring_check is supported for backward compatibility)
`iscobol.substring.zero_len_all (boolean) *`	**True =** a reference modifier with length = 0 behaves as if length was omitted (e.g. src-item(2:0) behaves like src-item(2:), so that the content of src-item from byte 2 to the end is returned). **False =** a reference modifier with length = 0 returns a variable whose length is 0 (e.g. src-item(2:0) returns ""). The default value is True.
`iscobol.switches *`	This property sets the switches as a sequence of numbers separated by commas. For example, if the program contains 10 switches and you wish to activate the first two and the fifth, set iscobol.switches=1,2,5 If the the switch name is a letter, such as switch "A"... switch "C"... switch "Z"... Then you need to specify the ordinal number of the letter in order to activate the corresponding switch, e.g. iscobol.switches=1,3,26.

Property	Meaning
iscobol.terminal.info.name iscobol.terminal.info.reverse iscobol.terminal.info.blink iscobol.terminal.info.underline iscobol.terminal.info.dual_intensity iscobol.terminal.info.132column iscobol.terminal.info.color iscobol.terminal.info.drawing iscobol.terminal.info.screen.lines iscobol.terminal.info.screen.columns iscobol.terminal.info.printer iscobol.terminal.info.attributes iscobol.terminal.info.graphic iscobol.terminal.info.screen.usable.height iscobol.terminal.info.screen.usable.width iscobol.terminal.info.screen.physical.height iscobol.terminal.info.screen.physical.width	These properties set the values returned in the corresponding data items of the TERMINAL-ABILITIES group item.
iscobol.terminal.info.refresh_monitor (boolean) *	**True =** inquire the system to obtain the screen resolution each time TERMINAL-INFO is accepted from TERMINAL-ABILITIES. **False =** inquire the system to obtain the screen resolution only the first time TERMINAL-INFO is accepted from TERMINAL-ABILITIES. Return the stored values to the next ACCEPT statements. The default value is False.

Property	Meaning

`iscobol.tracelevel`	This property allows the user to define the events to be traced. Valid values (which can be added together) are:
	1 = Settings of configuration properties and JVM options. Configuration properties set in the external environment are not traced in the logfile. Only the properties found in the configuration file, the properties set in the runtime command-line and the properties set by the programs are traced. The list of processed configuration files is also traced. For security reasons, the value of these settings is shown as encrypted in the log file:
	`iscobol.file.encryption.key`
	`iscobol.file.index.password`
	`iscobol.jdbc.password`
	`iscobol.net.ssl.key_store_password`
	`iscobol.net.ssl.trust_store_password`
	`iscobol.print.attribute.owner_password`
	`iscobol.print.attribute.user_password`
	`iscobol.sqlserver.password`
	`iscobol.user.password`
	2 = Program starts and program ends.
	4 = Paragraph starts and paragraph ends as well as method starts and method ends (program must be compiled with *-d* or *-dx*).
	8 = File I/O activities.
	16 = Content of keys (works only in conjunction with 8).
	32 = Content of record (works only in conjunction with 8).
	64 = Client / Server activity in terms of connection and disconnection (isCOBOL Server environment).
	128 = RPC calls for communication between client and server (isCOBOL Server environment).
	256 = SQL activity.
	512 = Complementary information to the one shown by pressing the Alt+Pause keyboard combination.
	1024 = Library routine starts and library routine ends. It works only in conjunction with the value 2.
	2048 = Internal information useful to programmers. This value activates the following traces:
	• ESQL cursors life cycle
	• Errors in the loading of fonts specified by *iscobol.font* settings
	
	The default value is 0.
	
	To produce a trace log, set iscobol.tracelevel to a non-zero value. With the default value of 0, the system does not create a log since it would be empty.
	
	The following are some useful settings:
	
	iscobol.tracelevel=3 includes config settings and program starts and ends.
	iscobol.tracelevel=7 includes config, program starts/ends, and paragraph starts/ends.
	iscobol.tracelevel=11 includes config, program starts/ends and file i/o (i.e. everything except for paragraph starts/ends).
	iscobol.tracelevel=43 includes config, program starts/ends and file i/o including the content of read records.
	iscobol.tracelevel=15 includes config, program and paragraph starts/ends, and file i/o.
	iscobol.tracelevel=63 traces everything of the above.
	
	You can add custom information to the logfile by calling the C$WRITELOG routine.
	
	The log file name is controlled by the property iscobol.logfile. If the property is not set, then a file named "iscobol0.log" is created in the temp folder.
	
	When the trace level is greater than zero, the stack of blocking exceptions is captured in the log in addition to being displayed on the user interface.

Property	Meaning
`iscobol.upper_lower_method`	This property defines the strategy used by the Framework in order to convert a string either to upper case or to lower case. Possible values are: **1 =** use the methods toUpperCase() and toLowerCase() of the java.lang.String class **2 =** use the methods toUpperCase() and toLowerCase() of the java.lang.Character class The default value is 1. In most cases, the result is the same. A difference may occur for some special characters; for example, the symbol "ß" is translated to "SS" when made upper case with method 1.
`iscobol.use_for_debugging (boolean) *`	**True =** Enable the USE FOR DEBUGGING declarative for programs compiled with the -cv option. **False =** The USE FOR DEBUGGING declarative is ignored. The default value is False.
`iscobol.utf16.little_endian (boolean) *`	**True =** The content of PIC N items is stored in UTF-16 Little Endian encoding. **False =** The content of PIC N items is stored in UTF-16 Big Endian encoding. The default value is False. This property is useful only to pass UTF-16 data to C functions, according to the endianness expected by the C function.

RemoteCompiler Configuration

RemoteCompiler properties cannot be set by SET ENVIRONMENT within the program. They must appear in the external configuration.

Property	Meaning
`iscobol.remotecompiler.compileonserver (boolean)`	**True =** After precompiling, the RemoteCompiler compiles the translated cbl to class on the server and sends the resulting class along with the translated cbl to the client. **False =** After precompiling, the RemoteCompiler sends the translated cbl to the client. The default value is False.
`iscobol.remotecompiler.conf`	This property specifies the name of the RemoteCompiler configuration file.
`iscobol.remotecompiler.createerrorfiles (boolean)`	**True =** The RemoteCompiler will ask preprocessors to create error files. **False =** No error files will be created by preprocessors. The default value is False.
`iscobol.remotecompiler.createlistingfiles (boolean)`	**True =** The RemoteCompiler will ask preprocessors to create listing files. **False =** No listing files will be created by preprocessors. The default value is False.
`iscobol.remotecompiler.host`	This property specifies the name of the server where the RemoteCompiler is active and listening. The default value is 'localhost'.

Property	Meaning
`iscobol.remotecompiler.port`	This property specifies the port on the server where the RemoteCompiler is listening. The default value is 11999.
`iscobol.remotecompiler.preprocnames`	This property specifies the precompilers that will run server side during a remote compilation. Multiple names must be separated by commas. The special value "ALL" can be used to instruct the RemoteCompiler to execute every preprocessor defined in its configuration. The same effect is obtained by omitting this property. The special value "NONE" can be used to avoid pre-compiling on the server machine and perform a standard COBOL compilation only.
`iscobol.remotecompiler.translateddir`	This property specifies an alternate folder in which to store translated files. By default they're stored in the same folder as the original source file.

WebDirect Configuration

WebDirect properties cannot be set by SET ENVIRONMENT within the program. They must appear in the external configuration.

Property	Meaning
`iscobol.wd2.additional_javascript`	This property allows users to load additional java scripts to be used in the web application. The js files specified in the property are loaded during startup, and are available throughout the application. The js files must be placed in the resources/js subfolder of the web application, and only their file name, complete with extension, must be specified as value of the property (i.e.: *iscobol.wd2.additional_javascript=script1.js*). Multiple values must be separated by \n or by the system path separator (i.e.: *iscobol.wd2.additional_javascript=script1.js\nscript2.js*).
`iscobol.wd2.additional_stylesheet`	This property allows users to load additional CSS stylesheets to be used in the web application. The CSS files specified in the property are loaded during startup, and are available throughout the application. The stylesheet files must be placed in the resources/css subfolder of the web application, and only their file name, complete with extension, must be specified as value of the property (i.e.: *iscobol.wd2.additional_stylesheet=mycustom.css*). Multiple values must be separated by \n or by the system path separator (i.e.: *iscobol.wd2.additional_stylesheet=mycustom.css\nmycustom2.css*).
`iscobol.wd2.doubleclick_speed`	This property specifies the amount of milliseconds between two mouse clicks to consider them a double click. The default value is 300.
`iscobol.wd2.style`	This property allows basic styling for controls in the application. Changing the values will cause runtime-generated CSS stylesheets to be created with different parameters, if no values are assigned to CSS-BASE-STYLE-NAME or to CSS-STYLE-NAME. Allowed values are: • **bs** = Bootstrap styling (see http://getbootstrap.com/ for details) • **default** = default styling • **os** = OS styling • **trendy** = Trendy styling These values refer to ZK embedded styles. The default value is *default*.

Property	Meaning
`iscobol.wd2.wait_message`	This property specifies the wait message that is shown by WebDirect during processing. The default value is: "Loading... Please wait until this screen is completely loaded."
`iscobol.wd2.mobile_numpad (boolean)`	This property takes effect when a WebDirect application is used from a mobile device. **True** = The numpad is shown instead of the standard keyboard when a numeric input is required. **False** = The standard keyboard is shown for every kind of input. The default value is False.

HTTPClient Configuration.

Property	Meaning
`iscobol.http.ignore_certificates (boolean) *`	**True** = If the handshaking fails due to invalid certificates, continue and connect anyway. **False** = If the handshaking fails due to invalid certificates, stop. The default value is False. See Connecting to a SSL-enabled web service for more information. Note - this property should be set to true only for test purposes. It's not good practice to ignore handshaking errors.
`iscobol.httpclient.logfile`	This property specifies the pathname of the log file where the HTTPClient activity is traced if iscobol.httpclient.logging (boolean) is set to true. Relative pathnames are resolved according to the runtime working directory. The log includes the request date and time, the request content, the response code, the response headers and the response content. If the file already exists, the new content is appended to it.
`iscobol.httpclient.logging (boolean)`	**True** = Trace the HTTPClient activity. **False** = Don't trace the HTTPClient activity. The default value is False.

HTTPHandler Configuration

(*) The asterisk after the property name means that the property is static, it's inquired only once by the Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.http.cgi_clear_missing_values (boolean) *`	**True** = Set the value of numeric data items to zero and non-numeric data items to spaces if a CGI variable is empty or does not exist. **False** = Don't clear the value of data items if a CGI variable is empty or does not exist. The default value is True.

Property	Meaning
`iscobol.http.cgi_content_type *`	This property specifies the MIME type of the CGI output. The default value is "text/html".
`iscobol.http.cgi_no_cache (boolean) *`	**True =** "Pragma: no-cache" is added to the HTTP response header. **False =** "Pragma: no-cache" is not added to the HTTP response header. The default value is True.
`iscobol.http.cookies_as_fields (boolean) *`	**True =** All the cookies can be read through the accept() method as if they were input fields. The cookie name is case sensitive. **False =** Cookies can't be read through the accept() method. The default value is False.
`iscobol.http.form.encoding *`	This property specifies the character set used by HTML forms that send data through HTTP (e.g. UTF-8). If not set, the framework tries to calculate the current character set itself.
`iscobol.http.html_template_prefix *`	This property lists the paths which HTML files are searched by *HTTPHandler:>processHtmlFile()*. Multiple values must be separated by the line feed character or by the current operating system path separator. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a".
`iscobol.http.mtom_enabled (boolean) *`	**True =** MTOM (Message Transmission Optimization Mechanism) enabled: a receiver MUST accept both a non-optimized and an optimized message, and a sender MAY send an optimized message, or a non-optimized message. The heuristics used by a sender to determine whether to use optimization or not are implementation-specific. **False =** MTOM (Message Transmission Optimization Mechanism) disabled The default value is False.
`iscobol.http.servlet.prefix`	This property specifies the prefix to put before the program name in the URL in order to build the name of the actual COBOL program. For example, given the following URL: *http://localhost:8080/myservlet/servlet/isCobol(PROG)* , if *iscobol.http.servlet.prefix=soap*, the framework will run the program SOAPPROG instead of PROG.
`iscobol.http.stateless (boolean) *`	**True =** Programs are automatically cancelled from memory when the main program returns (stateless). **False =** Programs remain in memory unless explicitly cancelled by a CANCEL statement (stateful) The default value is False.
`iscobol.http.upload.directory *`	This property specifies the directory where files uploaded through HTTP must be stored. A HTTP error is returned if the user tries to upload a file whose size exceeds the value of this property. By default, the user temp directory is used.
`iscobol.http.upload.max_size *`	This property specifies the maximum size in bytes allowed for file upload through HTTP. The default value is 1048576.

Property	Meaning
`iscobol.http.upload.prefix *`	This property specifies an optional prefix that must be applied to the name of the files uploaded through HTTP.
`iscobol.http.value_prefix_colon (boolean)`	**True** = Embedded values in the HTML code are expected to be prefixed by colon, E.g. *<html>hello :myvar world</html>*. **False** = Embedded values in the HTML code are expected to be enclosed by double percent sign, E.g. *<html>hello %%myvar%% world</html>*. The default value is False.
`iscobol.rest.default_stream`	This property specifies the default Content-type to be used when Content-type is not available in the HTTP header. It is considered by the HTTPHandler methods acceptEx and displayEx as well as the HTTPClient methods doPostEx and getResponseEx. The possible values are "xml" or "json". The default value is "json".
`iscobol.rest.log (boolean)`	**True** = Enables logging of the REST activity at global level. **False** = No log of the REST activity is performed at global level. The default value is False.
`iscobol.rest.log.<methodName> (boolean)`	**True** = Enables logging of the REST activity for the specified method name . **False** = No log of the REST activity is performed the specified method name . The default value is False.
`iscobol.rest.log.folder`	This property specifies the folder where REST activity log files are generated. The log file name is generated dynamically using the following pattern: *{methodname}-{SESSIONID}*.log. Backslashes must be doubled. For example, the path "C:\MyLogDir" would be "C:\\MyLogDir". The isCOBOL framework uses the java.util.logging package, and there are many configuration options. For example, you can specify "%h" in the iscobol.logfile and it will be replaced by the user's home directory. You can specify %yyyy, %mm, %dd, %hh, %nn, %ss and %cc and they will be replaced with the current year, month, day, hour, minute, second and hundredth of second respectively.
`iscobol.soap.log (boolean)`	**True** = Enables logging of the SOAP activity at global level. **False** = No log of the SOAP activity is performed at global level. The default value is False.
`iscobol.soap.log.<methodName> (boolean)`	**True** = Enables logging of the SOAP activity for the specified method name . **False** = No log of the SOAP activity is performed the specified method name . The default value is False.

Property	Meaning
`iscobol.soap.log.folder`	This property specifies the folder where SOAP activity log files are generated. The log file name is generated dynamically using the following pattern: *{methodname}-{SESSIONID}*.log. Backslashes must be doubled. For example, the path "C:\MyLogDir" would be "C:\\MyLogDir". The isCOBOL framework uses the java.util.logging package, and there are many configuration options. For example, you can specify "%h" in the iscobol.logfile and it will be replaced by the user's home directory. You can specify %yyyy, %mm, %dd, %hh, %nn, %ss and %cc and they will be replaced with the current year, month, day, hour, minute, second and hundredth of second respectively.

`iscobol.soap.wsdl.location`	This property specifies the location of the wsdl file. It should point to a file system path where wsdl files are copied, ie:
	
	`iscobol.soap.wsdl.location=/path/to/wsdl`
	
	Both full paths and relative paths are allowed. Relative paths are relative to the servlet container service working directory.
	
	The servlet appends the webservice name and ".wdsl" to this path to form a path name, which, if found, is then downloaded.
	If the file is not found or the property is not set, and HTTP error 404 is returned.
	
	The download can be achieved using an url such as:
	
	`http://localhost:8080/test/servlet/SONGS?wsdl`

User Interface Configuration

(*) The asterisk after the property name means that the property is static, it's inquired only once by the Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.accept_timeout`	This property causes all ACCEPT and DISPLAY MESSAGE BOX statements to time out just as if the BEFORE TIME clause was present. The property value specifies the timeout period, in seconds. This timeout is applied to every statement that can have a BEFORE TIME clause specified for it unless such clause has already been explicitly coded for the statement. The default value is 0, meaning no timeout value will be applied. When this property is set every ACCEPT is performed on a window and the runtime will no longer be able to ACCEPT user input from the console.
`iscobol.auto_input_mode (boolean) *`	**True =** The IME (Input Method Editor) activates itself automatically when the focus moves to a input field associated to a national data item. **False =** The IME (Input Method Editor) doesn't activate itself automatically when the focus moves to a input field associated to a national data item. It's user duty to activate the IME before inputting data. This property has effect when the active keyboard supports IME (for example if the active keyboard is Japanese) and it affects both graphical screens and character based screens. The default value is False
`iscobol.background_intensity *`	This property defines the default intensity of the background color of the windows. Valid values are: **0** = The default intensity for the output device is used. **1** = The background intensity is forced to low. **2** = The background intensity is forced to high. The default value is 0.
`iscobol.bitmap_scale.best_quality (boolean)`	**True =** The runtime tries to obtain the best quality during the scale of a bitmap, even if the process may require more time. **False =** The runtime scales bitmaps without particular optimizations, completing the process in the shortest time possible. This property affects the W$SCALE routine and the Bitmap-Scale property of the Bitmap control. The default value is False
`iscobol.colormap.default *`	This property associates a color for DISPLAY without any attributes. The value is the sum between *ForegroundColor* and *BackgroundColor* described in Using standard COBOL values. For example if you want a blue foreground on a white background, set *iscobol.colormap.default=258* where 258 is the result of the sum between bckgrnd-white (256) and blue (2).

Property	Meaning
`iscobol.colormap.high *`	This property associates a color for DISPLAY with the HIGH attribute. The value is the sum between *ForegroundColor* and *BackgroundColor* described in Using standard COBOL values. For example if you want a bright blue foreground on a white background, set *iscobol.colormap.high=266* where 266 is the result of the sum between bckgrnd-white (256) and bright-blue (10).
`iscobol.colormap.low *`	This property associates a color for DISPLAY with the LOW attribute. The value is the sum between *ForegroundColor* and *BackgroundColor* described in Using standard COBOL values. For example if you want a blue foreground on a white background, set *iscobol.colormap.low=258* where 258 is the result of the sum between bckgrnd-white (256) and blue (2).
`iscobol.font.<fontname>.cell *`	This property specifies a custom cell size associated with the font <fontname>. The runtime does not calculate the cell size as usual but uses the values specified by the user through the property instead. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels. <fontname> identifies a font by name, state and size. Upper-case characters are allowed here. State and size are optional and must be separated by dash if specified. If the font name contains spaces, then the \ character must be used to escape the spaces when the setting is done in the configuration file, while the whole setting must be enclosed between quotes if the setting is done in the command line. For example, if you want to force the cell size 10x10 for the Courier New font in bold state with size 12, you will write the following entry in the configuration file: `iscobol.font.Courier\ New-bold-12.cell=10,10` Each time the program loads the above font with the W$FONT routine and uses it on a graphical window, the custom cell size will be used.
`iscobol.font.default *`	This property specifies the font name to be used for the quick-loaded font "DEFAULT". The value format is: FontName-FontStyle-FontDim • *FontName* is the name of the font. The default value is Sans Serif. • *FontStyle* is the style of the font such as bold, italic, or bolditalic. • *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard default font (Sans Serif); no error is raised.
`iscobol.font.default.cell *`	This property specifies a custom cell size for the default font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.
`iscobol.font.fixed *`	This property specifies the font name to be used for the quick-loaded font "FIXED". The value format is: FontName-FontStyle-FontDim • *FontName* is the name of the font. The default value is Monospaced. • *FontStyle* is the style of the font such as bold, italic or bolditalic. • *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard fixed font (Monospaced); no error is raised.

Property	Meaning
`iscobol.font.fixed.cell *`	This property specifies a custom cell size for the fixed font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.
`iscobol.font.handling *`	This property controls the aliasing applied to fonts. The value of this property identifies a hint that the runtime passes to the underlying Swing control. If the Swing control honors the hint, you will notice a different font aliasing. Possible values for this property are: **0 =** The hint ANTIALIAS_DEFAULT is used for all controls. **1 =** The hint ANTIALIAS_ON is used for all controls **2 =** The hint ANTIALIAS_OFF is used for all controls **3 =** The hint ANTIALIAS_OFF is used for Frame, List-Box and Grid controls, as it happened in version 2013 R2 and previous. The default value is 0.
`iscobol.font.large *`	This property specifies the font name to be used for the quick-loaded font "LARGE". The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font. The default value is SansSerif. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard large font (Sans Serif); no error is raised.
`iscobol.font.large.cell *`	This property specifies a custom cell size for the large font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.
`iscobol.font.medium *`	This property specifies the font name to be used for the quick-loaded font "MEDIUM". The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font. The default value is SansSerif. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard medium font (Sans Serif); no error is raised.
`iscobol.font.medium.cell *`	This property specifies a custom cell size for the medium font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.
`iscobol.font.small *`	This property specifies the font name to be used for the quick-loaded font "SMALL". The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font. The default value is SansSerif. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard small font (Sans Serif); no error is raised.
`iscobol.font.small.cell *`	This property specifies a custom cell size for the small font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.

Property	Meaning
`iscobol.font.traditional *`	This property specifies the font name to be used for the quick-loaded font "TRADITIONAL". The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font. The default value is Monospaced. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the standard traditional font (Monospaced); no error is raised.
`iscobol.font.traditional.cell *`	This property specifies a custom cell size for the traditional font. The value format is *x,y* where *x* and *y* are the width and height of the cell in pixels.
`iscobol.foreground_intensity *`	This property defines the default intensity of the foreground color of the windows. Valid values are: **0** = The default intensity for the output device is used. **1** = The foreground intensity is forced to low. **2** = The foreground intensity is forced to high. The default value is 0.
`iscobol.gui.accept.before_time.repeat (boolean)`	**True =** The Accept timer is reset after each time the user inputs a new digit. **False =** The Accept timer stops as soon as the user inputs the first digit. The default value is False
`iscobol.gui.apply_window_color (booloean)*`	**True =** Control whose colors are not specified by the program inherit colors from the parent window. **False =** Control whose colors are not specified by the program inherit colors from the Look And Feel. The default value is True.
`iscobol.gui.column_separation`	This property specifies the default separation distance between columns in List-Box and Grid. The value is expressed in 10ths of characters. Default value is 5.

Property	Meaning
`iscobol.gui.Control.event`	This property specifies the value of Event-List property for all controls of class *Control* which Event-List property is not specified. This property has effect only in WebDirect environment and overrides iscobol.gui.events_list. *Control* can be one of the following: • bitmap • checkbox • combobox • dateentry • entryfield • grid • listbox • pushbutton • radiobutton • scrollbar • tab • treeview • webbrowser The value of this property is the list of numeric values of event constants as defined in the isgui.def copybook. Multiple values must be separated by comma.
`iscobol.gui.cscompress *`	This property allows you to activate the compression of data that's exchanged between the UI manager (client) and the code manager (server). It affects both stand alone and thin client executions. The server sends buffered data to the client when executing one of the following statements • ACCEPT • INQUIRE • SET INPUT WINDOW • SET I-O WINDOW • MODFIY *window-handle* VISIBLE|ENABLED when iscobol.gui.cstimeout * has expired or when the buffer size reaches the value specified by iscobol.gui.csmaxbuffersize *. The compression is performed only when the data size is equal or greater than the value specified by iscobol.gui.csminsizecompress *. Valid values are: **0** = no compression **1** = best compression **2** = fastest compression The default value is 0.

Property	Meaning
`iscobol.gui.csmaxbuffersize *`	This property sets the maximum size (in bytes) of the buffered data for the communication between the UI manager (client) and the code manager (server). It affects both stand alone and thin client executions. The server sends buffered data to the client when executing one of the following statements • ACCEPT • INQUIRE • SET INPUT WINDOW • SET I-O WINDOW • MODIFY *window-handle* VISIBLE|ENABLED when iscobol.gui.cstimeout * has expired or when the buffer is full. Setting this property to a lower value will result in more frequent updates to the user interface. A high value reduces the communication between client and server, but consumes more memory. A low value increases the communication between client and server, but saves memory. The default value is 1048576.
`iscobol.gui.csminsizecompress *`	This property specifies the minimum size in bytes for the compression of the client-server buffer activated by the property iscobol.gui.cscompress *. Only when this amount of bytes is reached the compression is applied. The default value is 32768.
`iscobol.gui.cstimeout *`	This property sets the timeout (in milliseconds) for the communication between the UI manager (client) and the code manager (server). It affects both stand alone and thin client executions. The server sends buffered data to the client when executing one of the following statements • ACCEPT • INQUIRE • SET INPUT WINDOW • SET I-O WINDOW • MODFIY *window-handle* VISIBLE|ENABLED when the buffer size reaches the value specified by iscobol.gui.csmaxbuffersize * or when the timeout specified by this property has expired. Setting this property to a lower value will result in more frequent updates to the user interface. A high value reduces the communication between client and server, but consumes more memory. A low value increases the communication between client and server, but saves memory. This property is automatically set to zero by the isCOBOL Debugger during the stand-alone debug. During the remote debug is your duty to set this property to zero, otherwise you might not see the result of a DISPLAY or MODIFY as soon as you step into the statement. Default value is 500.

Property	Meaning
`iscobol.gui.curr_bcolor`	This property specifies the background color of the current control. It affects ComboBox and EntryField controls. The runtime automatically applies the color when the control get focus and removes the color when the control loses focus. The property accepts numeric values of standard and RGB colors. See Color management for the list of possible values. The property is ignored by read-only fields. For example, if you want a bright blue background on the current field, set *iscobol.gui.curr_bcolor=9*, or, if you prefer to use RGB values, set *iscobol.gui.curr_bcolor=-255* (iscobol.gui.curr_ef_bcolor is supported for backward compatibility and it affects only EntryFields)
`iscobol.gui.curr_border_color`	This property specifies the border color of the current control. It affects only EntryField controls. The runtime automatically applies the color when the control get focus and removes the color when the control loses focus. The property accepts numeric values of standard and RGB colors. See Color management for the list of possible values. For example, if you want a bright blue border on the current field, set *iscobol.gui.curr_border_color=9*, or, if you prefer to use RGB values, set *iscobol.gui.curr_border_color=-255*
`iscobol.gui.curr_border_width`	This property specifies the width in pixels of the border of the current control. It affects only EntryField controls. The runtime automatically changes the border width when the control get focus and restores the original width when the control loses focus. This property should be set to a list of 4 numeric values separated by space that identify the top border, the left border, the bottom border and the right border widths respectively. For example, in order to make the top and bottom border bigger when the control gets the focus, set *iscobol.gui.curr_border_width=5 15 1*.
`iscobol.gui.curr_fcolor`	This property specifies the foreground color of the current control. It affects ComboBox and EntryField controls. The runtime automatically applies the color when the control get focus and removes the color when the control loses focus. The property accepts numeric values of standard and RGB colors. See Color management for the list of possible values. The property is ignored by read-only fields. For example, if you want a bright blue foreground on the current field, set *iscobol.gui.curr_fcolor=9*, or, if you prefer to use RGB values, set *iscobol.gui.curr_fcolor=-255* (iscobol.gui.curr_ef_fcolor is supported for backward compatibility and it affects only EntryFields)
`iscobol.gui.date_entry.cutoff`	This property establishes the two-digit years that will be interpreted by the program as being in the 20th Century and the two-digit years that will be interpreted by the program as being in the 21st Century. When set to a negative value, Java defaults are used. When set to a value greater than 99, only the last two digits of the value are considered. This setting affects a DateEntry with the Numeric style where a two-digit year instead of a four-digit year is input. The default value is -1.

Property	Meaning
`iscobol.gui.date_entry.century_date`	This property is used to set the default century date format for date-entry control. Default value depends on the current locale.
`iscobol.gui.date_entry.display_format`	This property is used to set the display format for date-entry control. Default value depends on the current locale.
`iscobol.gui.date_entry.errormessage`	This property specifies the text of the error message shown when a DateEntry validation fails. The DateEntry validation si activated by the property iscobol.gui.date_entry.validate (boolean).
`iscobol.gui.date_entry.long_date`	This property is used to set the default long date format for date-entry control. Default value depends on the current locale.
`iscobol.gui.date_entry.time`	This property is used to set the default time format for date-entry control. Default value is "HH:mm:ss".
`iscobol.gui.date_entry.validate (boolean)`	**True =** A date validation is automatically performed when the user leaves a DateEntry field. If the date is not valid, an error message is shown and the focus is kept on the field. You can configure the message text by setting iscobol.gui.date_entry.errormessage. **False =** No validation is automatically performed when the user leaves a DateEntry field. The default value is False.
`iscobol.gui.disabled_field_color`	This property specifies the color for disabled Entry-Field and Combo-Box. It can be set to a single value in order to specify a combined color or to two distinct values, separated by comma, in order to specify background-color and foreground-color. Negative values are considered RGB. For example, if you want that disabled fields are background blue and foreground white, you can set either *iscobol.gui.disabled_field_color=72* or *iscobol.gui.disabled_field_color=1,16* or *iscobol.gui.disabled_field_color=-128,-16777215* It is possible to append ";1" or ";0" to specify if this setting should override the COLOR property of the control: **0** = the color specified by this property is always applied. This is the default behavior. **1** = the color specified by this property is applied only if the COLOR property was not used on the Entry-Field. For example, if you only want disabled fields that don't have their own COLOR property to be set to background blue and foreground white, you can set either *iscobol.gui.disabled_field_color=72;1* or *iscobol.gui.disabled_field_color=1,16;1* or *iscobol.gui.disabled_field_color=-128,-16777215;1*

Property	Meaning
`iscobol.gui.ef.ext_message (boolean)`	**True =** An error message is shown when the user puts an alphabetic value into a numeric entry-field. **False =** A beep is played when the user puts an alphabetic value into a numeric entry-field. The default value is False.
`iscobol.gui.ef_lineseparator`	This property specifies the line separator character (or characters) used when returning the value of a MULTILINE entry-field. For example, if you wish to force the Windows line separator set *iscobol.gui.ef_lineseparator=\r\n* in the configuration file or use *SET ENVIRONMENT "gui.ef_lineseparator" TO x"0d0a"* in the COBOL program. The default value is the line feed character.
`iscobol.gui.implied_decimal (boolean)`	**True =** If no decimal separator is input by the user, a decimal separator is automatically applied by the Runtime according to the picture of the data-item bound to the input field. **False =** No decimal separator is automatically applied. This property affects both graphical entry fields and character based input fields. The default value is False. *iscobol.gui.entryfield.implied_decimal* is still supported for backward compatibility.
`iscobol.gui.entryfield.notify_change_delay *`	This property specifies how many milliseconds the runtime has to wait before firing a NTF-CHANGED event when the user changes the content of an Entry-Field. The Entry-Field property Notify-Change-Delay has priority over this setting. The default value is 0.

Property	Meaning

`iscobol.gui.entryfield.read_only_color`	This property specifies the color for read-only Entry-Field. It can be set to a single value in order to specify a combined color or to two distinct values, separated by comma, in order to specify background-color and foreground-color. Negative values are considered RGB. For example, if you want that read-only fields are background blue and foreground white, you can set either
	`iscobol.gui.entryfield.read_only_color=72`
	or
	`iscobol.gui.entryfield.read_only_color=1,16`
	or
	`iscobol.gui.entryfield.read_only_color=-128,-16777215`
	
	It is possible to append ";1" or ";0" to specify if this setting should override the COLOR property of the control:
	0 = the color specified by this property is always applied. This is the default behavior.
	1 = the color specified by this property is applied only if the COLOR property was not used on the Entry-Field.
	For example, if you only want read-only fields that don't have their own COLOR property to be set to background blue and foreground white, you can set either
	`iscobol.gui.entryfield.read_only_color=72;1`
	or
	`iscobol.gui.entryfield.read_only_color=64,16;1`
	or
	`iscobol.gui.entryfield.read_only_color=-255,-16777215;1`

`iscobol.gui.entryfield.read_only_cursor_arrow (boolean)`	**True =** Moving the mouse over a read-only field, the mouse shape remains an arrow **False =** Moving the mouse over a read-only field, the mouse shape changes to a vertical bar The default value is False.
`iscobol.gui.entryfield.spell_checking_delay`	This property specifies the delay in milliseconds for the tool-tip shown by the spell checking feature of the Entry-Field control. The default value is 500.
`iscobol.gui.events_list`	This property specifies the value of the Event-List property for all controls for which the Event-List property is not specified. The value of this property is the list of numeric values of event constants as defined in the isgui.def copybook. Multiple values must be separated by comma. In a WebDirect environment this property may be overridden by iscobol.gui.Control.event.
`iscobol.gui.exclude_events (boolean)`	**True =** All controls for which the Exclude-Event-List property is not specified assume EXCLUDE-EVENT-LIST=1 **False =** All controls for which the Exclude-Event-List property is not specified assume EXCLUDE-EVENT-LIST=0 The default value is False.

Property	Meaning
iscobol.gui.fields_unboxed (boolean)	**True** = Entry field controls that do not have boxed style or 3-D style set use no-box style by default. **False** = Entry fields controls that do not have boxed style or 3-D style set use box style by default. The default value is True. (iscobol.fields_unboxed is supported for backward compatibility)
iscobol.gui.input_predisplay (boolean) *	**True** = During Accept the runtime has to refresh the values of the variables defined in SCREEN SECTION. **False** = During Accept the runtime doesn't refresh the values of the variables defined in SCREEN SECTION. The default value is False. (iscobol.gui.from_fields_refreshed_in_accept is supported for backward compatibility)
iscobol.gui.grid.extended_finish_reason (boolean) *	**True** = Additional Finish-Reason values are returned by the MSG-FINISH-ENTRY event in Grid. **False** = Standard Finish-Reason values are returned by the MSG-FINISH-ENTRY event in Grid. The default value is False.
iscobol.gui.grid.find_delay	This property specifies when the grid should start filtering data while the user interacts with the search panel. Possible values are: **0** - wait for the user to press Enter or click the Find button before start searching. **>0** - wait this number of milliseconds before start searching automatically. If the user presses Enter or clicks the Find button before this timeout expires, start searching. The default value is 500.
iscobol.gui.grid.lm_on_columns (boolean) *	**True** = Grid columns of grids with the Adjustable-Columns style are automatically resized when the window is resized and a layout manager is involved. **False** = Grid columns are not affected by layout managers. The Grid property Lm-On-Columns has priority over this setting. The default value is True.
iscobol.gui.grid.no_cell_drag (boolean) *	**True** = Grids don't fire drag events. **False** = Grids fire drag events. The Grid style No-Cell-Drag has priority over this setting. The default value is False.

Property	Meaning
iscobol.gui.hints_off	This property sets the amount of time, in milliseconds, that elapses before the hints are removed from the screen. (iscobol.hints_off is supported for backward compatibility)
iscobol.gui.hints_on	This property sets the amount of time, in milliseconds, that elapses before the hints are displayed. (iscobol.hints_on is supported for backward compatibility)
iscobol.gui.kbd_case	This property defines how characters input by the user are converted into GUI screen fields. Possible values are: **Upper** = All characters are converted uppercase **Lower** = All characters are converted to lowercase **Both** = All characters are entered as typed by user The default value is Both. Note - This setting may be overridden with Screen Section styles like Entry-Field's UPPER and LOWER or Grid's DATA-TYPES.
iscobol.gui.keyboard_buffering	The isCOBOL Framework bufferizes keyboard events in order to avoid losing input digits between different Accepts on different windows. This keyboard buffering can be configured by setting this property to one of the following values: **-1** = no limit on buffering **0** = disable keyboard buffering **>0** = specifies the number of keyboard events that must be bufferized The default value is -1.
iscobol.gui.icon (boolean) *	**True** = The isCOBOL logo is displayed on the windows of the program. **False** = The Java logo is displayed on the windows of the program. This property is considered only if iscobol.gui.icon_file is not set or points to an invalid file. This property affects only windows for which the Icon property was not specified. The default value is True. (iscobol.icon is supported for backward compatibility)
iscobol.gui.icon_file	This property specifies an image file to be used as the default icon for windows. The following image formats are accepted: BMP, JPG, GIF, ICO and PNG. This property affects only windows for which the Icon property was not specified. If this property is not set or the file can't be loaded, then the windows icon is controlled by the property iscobol.gui.icon (boolean) *.
iscobol.gui.icons_scaling (boolean)	**True** = Check-Box and Radio-Button default icons are resized along with the font under the LM-ZOOM layout manager. **False** = Check-Box and Radio-Button default icons always have a fixed size. The default value is False.

Property	Meaning
`iscobol.gui.ignore_invalid_handle (boolean)`	This property affects the way the Runtime Framework behaves when an INQUIRE or MODIFY statement affects an invalid control handle. **True =** No error is returned. **False =** An "Invalid handle" message is shown and the program terminates. The default value is False. (iscobol.ignore_invalid_handle is supported for backward compatibility)
`iscobol.gui.javabean.catch_exception (boolean)`	**True =** Exceptions generated by Java-Beans are printed on the system error. **False =** Exceptions generated by Java-Beans are hidden to the user. The default value is True.
`iscobol.gui.justify_num_fields (boolean)`	**True =** Numeric and edited fields are automatically right-aligned. **False =** Numeric and edited fields are left-aligned, unless otherwise specified. The default value is False. (iscobol.justify_num_fields is supported for backward compatibility)
`iscobol.gui.label.rtrim (boolean)`	**True =** Spaces on the right side of the title label are trimmed away. **False =** Spaces to the right of the title label are not removed. Default value is True.
`iscobol.gui.layout_manager`	When this property is set, the RESIZABLE style is assumed for all windows and all windows use the layout manager specified by the property, unless different settings were used in the DISPLAY WINDOW statement. Specifying a different layout manager or NOT RESIZABLE in the DISPLAY WINDOW statement invalidates the feature for that specific window. Possible values are: • lm-scale • lm-zoom
`iscobol.gui.layout_manager.max_font_zoom`	Specifies the higher limit for the font increase performed by LM-ZOOM. The value is expressed in percentage having 100 as the original font dimension. For example, having an original font size of 12 and setting this property to 150, when the user increases the window's height, LM-ZOOM will increase the font size accordingly until it reaches the size of 24. After it, the font size will not be increased anymore.
`iscobol.gui.layout_manager.min_font_zoom`	Specifies the lower limit for the font reduction performed by LM-ZOOM. The value is expressed in percentage having 100 as the original font dimension. For example, having an original font size of 12 and setting this property to 50, when the user reduces the window's height, LM-ZOOM will reduce the font size accordingly until it reaches the size of 6. After it, the font size will not be reduced anymore.

Property	Meaning
`iscobol.gui.light_gray_is_transparent (boolean)`	**True** = The RGB color 0xC0C0C0 in bitmaps is considered transparent color. **False** = The RGB color 0xC0C0C0 in bitmaps is preserved and rendered as gray. This propery affects all the images loaded by the W$BITMAP routine in both the graphical user interface and the print jobs. The default value is True.
`iscobol.gui.lightweightpopup (boolean)`	**True** = lists dropped by menu bar and combo-boxes as well as tool-tips are in the background and can be covered by a web-browser control on the screen. **False** = lists dropped by menu bar and combo-boxes as well as tool-tips are in the foreground and cannot be covered by any control. The default value is True.
`iscobol.gui.list.lm_on_columns (boolean)*`	**True** = ListBox columns are automatically resized when the window is resized and a layout manager is involved. **False** = ListBox columns are not affected by layout managers. The ListBox property Lm-On-Columns has priority over this setting. The default value is True.
`iscobol.gui.max_text_in_bytes (boolean)`	**True** = the text typed by the user in entry-fields and grids is measured in bytes, i.e. a double byte character counts as two. **False** = the text typed by the user in entry-fields and grids is measured in characters, i.e. a double byte character counts as one. The default value is True.
`iscobol.gui.menu.altkey_default (boolean)*`	**True** = menu bar items with no subitems require two steps in order to be selected by keyboard: press Alt+keyletter to activate the item and press Enter to select it (Java Swing behavior). **False** = menu bar items with no subitems are selected by keyboard by just pressing Alt+keyletter. The default value is True.

`iscobol.gui.messagebox.bcolor`	This property specifies the background that must be used by message boxes where the neither the BACKGROUND-COLOR clause nor the COLOR clause were specified in the DISPLAY MESSAGE BOX statement.
	
	Negative values are considered RGB. For example, if you want the background color to be blue, you can set either
	`iscobol.gui.messagebox.bcolor=1`
	or
	`iscobol.gui.messagebox.bcolor=-128`
	
	The default value is LAF dependent.

`iscobol.gui.messagebox.centered (boolean)`	**True** = The CENTERED clause is assumed for the DISPLAY MESSAGE BOX statement. **False** = The CENTERED clause is not assumed for the DISPLAY MESSAGE BOX statement. The default value is False.

Property	Meaning

`iscobol.gui.messagebox.custom_prog`	This property specifies the name of a COBOL program that will be automatically called whenever a DISPLAY MESSAGE BOX statement is used.
	
	The name can be followed by comma and a letter to specify the program location. When 'C' is used, the program is searched for in the client machine, otherwise it's searched for in the server machine.
	
	The program name and the next letter are case insensitive.
	
	Examples:
	Equivalent settings to load the program from the server's Classpath or code-prefix:
	`iscobol.gui.messagebox.custom_prog=MYMSG`
	`iscobol.gui.messagebox.custom_prog=MYMSG,S`
	Settings to load the program from the client's Classpath or code-prefix:
	`iscobol.gui.messagebox.custom_prog=MYMSG,C`
	
	See Custom message box implementation for more details.

`iscobol.gui.messagebox.fcolor`	This property specifies the foreground that must be used by message boxes where the neither the FOREGROUND-COLOR clause nor the COLOR clause were specified in the DISPLAY MESSAGE BOX statement.
	
	Negative values are considered RGB. For example, if you want the foreground color to be blue, you can set either
	`iscobol.gui.messagebox.fcolor=1`
	or
	`iscobol.gui.messagebox.fcolor=-128`
	
	The default value is LAF dependent.

`iscobol.gui.messagebox.font`	This property specifies the font that must be used by message boxes where the FONT clause was not specified in the DISPLAY MESSAGE BOX statement. The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. Example: *iscobol.gui.messagebox.font=Arial-Bold-08* The default value is LAF dependent.
`iscobol.gui.native_name`	**True =** the Screen Section name of controls is shown in the message box produced by pressing Alt+Pause. **False =** the internal name of controls is shown in the message box produced by pressing Alt+Pause. Setting this property to true makes isCOBOL export the COBOL name in the corresponding Java control. This may help when using JVM monitor tools. Default value is False.

Property	Meaning
`iscobol.gui.native_style (boolean) *`	**True** = a LAF dependent border is applied to all controls with a BOXED style (e.g. EntryFields, ComboBoxes, Grids...). It doesn't work on controls with the 3-D style. **False** = a standard black border is applied to all controls with a BOXED style unless the Border-Color property is set for them. The native style is more evident on ComboBoxes with the DROP-LIST style. Default value is False.
`iscobol.gui.nested_embedded_proc_check (boolean)*`	**True** = raise an exception for nested embedded procedures. **False** = don't raise any exception for nested embedded procedures. When the property is set to true, the "Nested Accept in embedded procedure." exception might be raised. Default value is False.
`iscobol.gui.ntf_resized_delay`	This property specifies the timeout in milliseconds before the runtime sends a NTF-RESIZED event to the program during window resizing. A value of 0 means that the event must be sent as soon as the resize occurs. A value greater than 0 will make the runtime wait. If another NTF-RESIZED event occurs before the timeout expires, the timer is restarted. When the timeout expires, one NTF-RESIZED event is sent to the program. Setting this property reduces the number of events generated during window resizing and improves performance. A reasonable value is 100. The default value is 0.

`iscobol.gui.placeholder_color`	This property specifies the placeholder text color for Entry-Field and Combo-Box. Negative values are considered RGB. For example, if you want the placeholder text to be blue, you can set either
	`iscobol.gui.placeholder_color=1`
	or
	`iscobol.gui.placeholder_color=-128`
	
	By default, the disabled field foreground color specified by the LAF is used as placeholder text color.

`iscobol.gui.push_activated_by_enter (boolean)`	**True** = Push-Buttons can always be activated by pressing Enter when they have the focus. **False** = Push-Buttons can be activated by pressing Enter when they have the focus only if the iscobol.key.enter setting includes "termination=13". The default value is False.
`iscobol.gui.click_override_focus_change (boolean)`	**True** = the click on Check-Box, Grid, Push-Button and Radio-Button is always intercepted even if the focus was forced on an input field by setting Screen-Control items in the field After Procedure. **False** = the click on Check-Box, Grid, Push-Button and Radio-Button is not intercepted if the focus was forced on an input field by setting Screen-Control items in the field After Procedure. The default value is True. (iscobol.gui.push_override_focus_change is still supported for backward compatibility but it affects only the Push-Button controls)

Property	Meaning
`iscobol.gui.quit_mode`	This property configures the behavior of the close button on the system menu of initial and standard windows. Independent and floating windows are not affected by this property. Possible values are: **-2 =** Clicking on the close button has no effect. **0 =** Clicking on the close button closes the window and the program terminates. **>0 =** Clicking on the close button raises an exception on the current ACCEPT. The CRT STATUS variable is set to the value of quit_mode. The default value is 0. (iscobol.quit_mode is supported for backward compatibility)
`iscobol.gui.rollover_border_color`	This property specifies the color that the border of boxed Entry-Fields must assume when the mouse hovers over the field. The property accepts numeric values of standard and RGB colors. See Color management for the list of possible values. For example, if you want a bright blue border when the mouse hovers over an Entry-Field, set *iscobol.gui.rollover_border_color=9*, or, if you prefer to use RGB values, set *iscobol.gui.rollover_border_color=-255*
`iscobol.gui.rollover_border_width`	This property specifies the width in pixels that the border of boxed Entry-Fields must assume when the mouse hovers over the field. This property should be set to a list of 4 numeric values separated by space that identify the top border, the left border, the bottom border and the right border widths respectively. For example, in order to make the top and bottom border bigger when the mouse hovers over an Entry-Field, set *iscobol.gui.rollover_border_width=5 15 1*.
`iscobol.gui.screen_col_plus_base`	This property controls the column adjustments when relative values are used by the program for the COLUMN property in the Screen Section. When the value is -1, "COLUMN + 0" and "COLUMN + 1" produce adjacent items. When the value is 0, "COLUMN + 0" produces adjacent items, and "COLUMN + 1" puts a space between items. The default value is -1.
`iscobol.gui.screen_col_zero (boolean)`	This property is evaluated each time a statement like "DISPLAY *foo* LINE *line-number* COL 0" occurs: **True =** *stuff* is displayed after the last item on line *line-number*. **False =** *stuff* is displayed at line *line-number* column 1. The default value is False.
`iscobol.gui.scrn_size_cols`	This property sets the SIZE of the default initial window in character-based environments (it doesn't affect graphical windows). The default value is 80.
`iscobol.gui.scrn_size_rows`	This property sets the LINES of the default initial window in character-based environments (it doesn't affect graphical windows). The default value is 25.

Property	Meaning
`iscobol.gui.show_zeroes (boolean) *`	**True** = leading zeroes are shown when displaying numeric data **False** = leading zeroes are not shown when displaying numeric data This property affects all the output devices: console, character-based terminals and graphical windows. The default value is False
`iscobol.gui.temporary_controls (boolean) *`	**True** = all graphical controls are TEMPORARY by default **False** = all graphical controls are PERMANENT by default The default value is False.
`iscobol.gui.tool_bar.native (boolean) *`	**True** = tool bars are implemented using the JToolBar class; in this way the Moveable style is supported, but buttons are always flat under the Windows look and feel. **False** = tool bars are implemented using the JPanel class; in this way you can have 3-D buttons under the Windows look and feel, but the Moveable style has no effect. The default value is True.
`iscobol.gui.treeview.selection_delay *`	This property sets the timeout in milliseconds for the TrreeView to start searching for an item while the user is typing. The user can select TreeView items by typing part of their name with the keyboard. The TreeView bufferizes the inputed digits and, when this timeout expires, it starts searching for the first matching item. A value of -1 makes the TreeView inherit this setting from the current LAF. The default value is -1.
`iscobol.gui.waitcursordelay`	In thin client environment, in case of slow server to client answers, the mouse pointer changes from default-cursor to wait-cursor. This property sets the timeout in milliseconds before the cursor is changed. A value of 0 disables the feature. The default value is 0.
`iscobol.gui.web_browser.home`	This property specifies the URL loaded when the GO-HOME Property of the WEB-BROWSER Control is set to a non-zero value. The default value is *https://www.veryant.com*. (iscobol.web_browser.home is supported for backward compatibility)
`iscobol.gui.web_browser.search`	This property specifies the URL loaded when the GO-SEARCH Property of the WEB-BROWSER Control is set to a non-zero value. The default value is *https://www.google.com*. (iscobol.web_browser.search is supported for backward compatibility)
`iscobol.gui.webbrowser.class`	This property specifies the class that provides the web-browser feature. Possible values are: • com.iscobol.browser.dj.DJWebBrowser • com.iscobol.browser.fx.JFXWebBrowser • com.iscobol.browser.jx.JXWebBrowser The default value is com.iscobol.browser.dj.DJWebBrowser.

Property	Meaning
`iscobol.gui.webbrowser.no_msg_before_navigate (boolean)`	**True** = No-Msg-Before-Navigate style is implicitly defined for all Web-Browser controls. **False** = No-Msg-Before-Navigate style is not implicitly defined for all Web-Browser controls. The default value is False
`iscobol.gui.window.auto_resize.fixed_dim (boolean)`	**True** = a window with the AUTO-RESIZE style can be resized only to a smaller dimension; it cannot be increased **False** = a window with the AUTO-RESIZE style can resized freely The default value is True
`iscobol.gui.windows_darkening *`	This property specifies the opacity of the parent window of modal floating windows and message boxes. A value of 0 means default opacity. A positive value ranging from 1 to 100 specifies the level of darkening with 1 representing an imperceptible darkening and 100 representing the black color. A negative value ranging from -1 to -100 specifies the level of transparency with -1 representing an imperceptible transparency and -100 representing the full transparency. The opacity is applied to windows when they're blocked by a modal dialog. Having iscobol.gui.windows_modality (boolean) * set to true (default), a window is blocked when a descendant modal dialog is opened. Having iscobol.gui.windows_modality (boolean) * set to false, a window is blocked when any modal dialog that is not its ancestor is opened. The default value is 0.
`iscobol.gui.windows_modality (boolean) *`	This property takes effect in scenarios where there are multiple initial windows. This is typical in applications that call multiple main programs using the CALL RUN statement. This property affects both Floating windows and message boxes. **True** = modal windows block all windows in the same hierarchy **False** = modal windows block all windows in the runtime session The default value is True
`iscobol.gui.window_title`	This property specifies the title of INITIAL/STANDARD windows displayed without the TITLE phrase. Without this setting, the name of the program is used as title of INITIAL/STANDARD windows displayed without the TITLE phrase.

Property	Meaning
`iscobol.gui.windows_uncropped (booloean)`	This property affects the position where child windows and message boxes are displayed. It can happen that, due the parent window position and the child window dimensions, the child window is cropped by the screen edges. By setting this property, you can instruct the runtime to alter the child window position to avoid the cropping. **True** = the runtime ensures that the bottom right border of the child window is not cropped by the screen edges. It moves the window if necessary. **False** = the runtime displays the child window in the position requested by the program, even if the window bottom right border is over the screen edges. The default value is True
`iscobol.guifactory.class *`	This property specifies the factory class that is used for GUI. Valid values are: • com.iscobol.gui.client.swing.GuiFactoryImpl • com.iscobol.gui.client.zk.GuiFactoryImpl • com.iscobol.gui.client.charva.GuiFactoryImpl The default value is com.iscobol.gui.client.swing.GuiFactoryImpl
`iscobol.message.cancel`	This property specifies the title of the "Cancel" button in the window invoked by the DISPLAY MESSAGE BOX Statement. The default value is "&Cancel".
`iscobol.message.no`	This property specifies the title of the "No" button in the window invoked by the DISPLAY MESSAGE BOX Statement. The default value is "&No"
`iscobol.message.ok`	This property specifies the title of the "OK" button in the window invoked by the DISPLAY MESSAGE BOX Statement. The default value is "&OK".
`iscobol.message.yes`	This property specifies the title of the "Yes" button in the window invoked by the DISPLAY MESSAGE BOX Statement. The default value is "&Yes".
`iscobol.win3_grid`	This property specifies a value defining the color of a grid displayed on all windows. This property has meaning only in a development environment. Its purpose is to help the programmer align controls in the window. Possible values are: **-1** = no color (feature disabled) **1** = black **2** = blue **3** = green **4** = cyan **5** = red **6** = magenta **7** = yellow Any other value will specify unpredictable colors. The default value is -1.

Property	Meaning
`iscobol.terminal.alpha_autoclear (boolean)`	**True =** clear the area as soon as the user inputs a digit. **False =** don't clear the area where the ACCEPT is performed. Keep the value on video and allow to overwrite the single digits. If affects ACCEPT with the UPDATE clause on alphanumeric fields. The default value is False.
`iscobol.terminal.antialiasing (boolean)`	**True =** antialiasing is applied to fonts on video. **False =** fonts on video are shown as they are. This property affects only character-based displays. Graphical controls always use antialiasing. The default value is False.
`iscobol.terminal.autowrap (boolean)`	**True =** DISPLAY statements longer than one line will wrap around on a character-based display. **False =** DISPLAY statements longer than one line will be truncated on a character-based display. The default value is False.
`iscobol.terminal.cl2end_fill_spaces (boolean)`	**True =** The field is filled with spaces from the cursor position until the end. If the AUTO clause was used or the program is compiled with the -va option, this causes the cursor to be automatically moved to the next input field or the termination of the accept if no more fields are available. **False =** The field is filled with spaces from the cursor position until the end, but the cursor position is unchanged. The default value is False. See Keyboard Configuration in order to know how to assign the cl2end function to a key.
`iscobol.terminal.cursor_blink`	This property configures the cursor blink for character based ACCEPT. Possible values are: **0** = no blink **1** = blink every 600 milliseconds **>1** = blink every n milliseconds, where n is the value of this property The default value is 0.

Property	Meaning
`iscobol.terminal.cursor_color`	This property configures the cursor color for character based ACCEPT. Possible values are: **-1** = default color **0** = black **1** = blue **2** = green **3** = cyan **4** = red **5** = magenta **6** = brown **7** = white **8** = dark gray **9** = bright blue **10** = bright green **11** = bright cyan **12** = bright red **13** = bright magenta **14** = yellow **15** = bright white The default value is -1.
`iscobol.terminal.cursor_type`	This property configures the cursor shape for character based ACCEPT. Possible values are: **0** = invisible **1** = underscore shape **2** = block shape **3** = underscore shape, becomes block while in insert mode **4** = vertical bar The default value is 2

`iscobol.terminal.data_range`	This property allows you to filter characters accepted in a character-based screen. The value format is *minVal[,maxVal]*, where minVal and maxVal are integer numbers representing the ASCII value of the character. If a character is not inside the range specified by the property, it is replaced with a space.
	For example, in order to skip all characters below the space, set
	
	`iscobol.terminal.data_range=32`
	
	In order to skip all characters that are not upper case letters, set
	
	`iscobol.terminal.data_range=65,90`

`iscobol.terminal.drag_enabled (boolean)`	**True =** the user can select an area in the window by dragging the mouse. On character-based screens the highlighted text is copied to the clipboard. **False =** dragging the mouse in the window doesn't produce a selection. The default value is True.

Property	Meaning
iscobol.terminal.edited_formatted (boolean)	**True =** numeric edited fields are not cleared when the first valid character is imputed and the editing characters are automatically skipped, except for the B editing character. Only the fields whose edited picture contains characters different from '9', 'Z', '+', and '-' are affected. **False =** numeric edited fields are cleared when the first valid character is imputed. The editing characters are removed as well during the cleaning. The default value is False.
iscobol.terminal.kbd_case	This property defines how characters input by the user are converted into character-based screen fields. Possible values are: **Upper =** All characters are converted uppercase **Lower =** All characters are converted to lowercase **Both =** All characters are entered as typed by user The default value is Both. Note - This setting may be overridden with the settings of the "UPPER" and "LOWER" keywords on individual ACCEPT statements. (iscobol.keyboard.kbd_case is still supported for backward compatibility)
iscobol.terminal.lines_3d (boolean)	**True =** lines and boxes whose color is black as rendered with a 3D effect. **False =** lines and boxes are rendered normally. The default value is False.
iscobol.terminal.no_autoclear (boolean)	**True =** don't clear the area where the ACCEPT is performed. **False =** clear the area where the ACCEPT is performed. If affects ACCEPT without the UPDATE clause. The default value is False.
iscobol.terminal.numeric_autoclear (boolean)	**True =** clear the area as soon as the user inputs a digit. **False =** don't clear the area where the ACCEPT is performed. Keep the numeric value on video and allow to overwrite the single digits. If affects ACCEPT with the UPDATE clause on numeric fields. The default value is True.
iscobol.terminal.screen_prompt	This property sets the character used for PROMPT for character-based ACCEPT. Default value:_
iscobol.terminal.screen_prompt_all	This property specifies where the PROMPT character must be shown. Possible values are: **NO -** the PROMPT character is shown only in the active field, where the cursor is. **YES -** the PROMPT character is shown in every field involved in the current ACCEPT. **PROTECTED -** the PROMPT character is shown in every field involved in the current ACCEPT except for protected fields. Default value: NO

Property	Meaning
`iscobol.terminal.update_from_screen (boolean)`	**True** = read content from screen during ACCEPT. **False** = don't read content from screen during ACCEPT. The default value is False.

Debugger Configuration

Debugger properties cannot be set by SET ENVIRONMENT within the program. They must appear in the external configuration.

Property	Meaning
`iscobol.debug.code_prefix`	The Debugger uses this property to locate the source code of a program. Set this property to the list of paths where source files can be found. Separate multiple paths by your system path separator. This property is considered in two conditions: - if the program was compiled with isCOBOL 2020 R1 or previous, or - if iscobol.debug.embedded_source (boolean) is set to false (iscobol.debug_code_prefix is supported for backward compatibility)
`iscobol.debug.embedded_source (boolean)`	**True** = The Debugger extracts the source code from the class file. **False** = The Debugger ignores the source code in the class file and looks for the source code on disk. The source code is searched in the following places: • the working directory that was used by the Compiler command • the paths and libraries listed in the Classpath • the paths listed by the `iscobol.debug.code_prefix` property The default value is True.
`iscobol.debug.port`	This property specifies the port used by the Remote Debugger. It can be used in the Framework configuration in order to tell on which port it should listen from connections. It can be used in the Debugger configuration in order to tell on which port to connect instead of passing this information on the command line. There is no point in using this property in the isCOBOL Server configuration for thin client debugging, as the isCOBOL Client automatically asks the isCOBOL Server to listen for Debugger connections on a specific port, according to the -debugport option (see Format 6 of isCOBOL Client usage). The default value is 9999.
`iscobol.debug.propfile`	This property specifies the name of the file where the Debugger saves and loads settings. Absolute and relative file names are allowed. By default settings are saved in a file named *isdebugger.properties* under the user home directory.

Property	Meaning
`iscobol.rundebug.redirect_streams (boolean)`	The property affects only the remote debugging of a stand-alone program launched with a command like *java -Discobol.rundebug=2 -Discobol.rundebug.redirect_streams=0 PROGRAM_NAME*. It doesn't work in thin client environment or with the stand-alone Debugger. **True** = The stdin stream is managed by the Debugger. When an ACCEPT FROM SYSIN is performed, you have to input the data in the Debugger's Command Area. **False** = The stdin stream is managed by the Runtime. When an ACCEPT FROM SYSIN is performed, you have to input the data in the Runtime console. This is particularly useful if the input is passed through a text file on the command line (e.g. *iscrun MYPROG < input.txt*). The stdout and stderr streams are caught by the Debugger's Console tab, as long as the Console is attached. If the Console is detached, then stdout and stderr are displayed on the Runtime console output. The default value is True. (iscobol.redirect_streams is supported for backward compatibility)
`iscobol.rundebug.stop_immediately (boolean)`	**True** = When debugging a remote application, you enter in debug as soon as a program compiled in debug mode is loaded. **False** = When debugging a remote application, you enter in debug only when a breakpoint is reached or the Pause button is pressed on the Graphical Debugger's tool-bar. The default value is True.

File Handling Configuration

The following configuration settings affect COBOL i-o statements. Library routines are not affected unless otherwise specified in the routine documentation.

(*) The asterisk after the property name means that the property is static, it's inquired only once by the Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.ctree.bound_library *`	Specifies the name of the c-tree server main library. The default value is "ctdbsapp".
`iscobol.ctree.bound_server (boolean)*`	**True** = c-tree works in stand-alone mode. This mode supports one client process only, so it is suggested that it is used only in an Application Server environment or single-user installations. **False** = c-tree works in client/server mode. The default value is False. (iscobol.ctree.ace is supported for backward compatibility)

Property	Meaning
`iscobol.ctree.new_config (boolean)*`	**True** = The Runtime Framework searches for the c-tree configuration between Framework Properties. See Configuring the client through Framework properties for the list of available properties. **False** = The Runtime Framework searches for the c-tree configuration in CTREE_CONF. The default value is True
`iscobol.extfh.keep_trailing_spaces (boolean)`	**True** = Sequential files managed through EXTFH preserve trailing spaces at the end of the record. **False** = Sequential files managed through EXTFH strip trailing spaces at the end of the record. The default value is True.
`iscobol.extfh.intrinsic_file_manager (boolean)`	**True** = "EXTFH input" does not use the "isCOBOL FileManager chooser" but rather uses the "Default isCOBOL File manager" **False** = "EXTFH input" uses the "isCOBOL FileManager chooser", so it uses the file manager specified by iscobol.file properties (e.g. indexed files will be managed by th e handlers specified by iscobol.file.index and iscobol.file.index.FileName). The default value is True.
`iscobol.extfh.libname`	Specifies the name of the EXTFH library
`iscobol.apply_code_path (boolean)`	**True** = The code_prefix is also used for programs whose name begins with "/" or "\". **False =** The code_prefix is used only for programs with a relative path name. For example, having *iscobol.code_prefix=/myapp* and loading "/obj/SUB1", the class searched by the runtime depends on the apply_code_path setting. When the setting is True, the runtime will look for "/myapp/obj/SUB1". When the setting is False, the runtime will look for "/obj/SUB1". The default value is False.
`iscobol.file.apply_file_path (boolean)`	**True** = The file_prefix is also used for files whose name begins with "/" or "\". **False =** The file_prefix is used only for files with a relative path name. For example, having *iscobol.file.prefix=/myapp* and opening "/data/FILE1", the file searched by the runtime depends on the apply_file_path setting. When the setting is True, the runtime will look for "/myapp/data/FILE1". When the setting is False, the runtime will look for "/data/FILE1". The default value is False. (iscobol.apply_file_path is supported for backward compatibility)

Property	Meaning
iscobol.file.binary_file_prefix	This property lists the paths in which to search unencoded sequential files (files whose names start with + characters). When a data file is opened for input or i-o, the Framework looks for the data file in every path listed by this property until it finds the data file. If the data file is not found, an error is returned. When the data file is opened for output, the Framework looks for the data file in every path listed by this property until it finds the data file, then it initializes it. If the data file is not found, then the Framework tries to create it in the first path listed by this property. Paths must be separated by the a line feed character or by the current operating system path separator. If one of the paths starts with "isf://", then paths must be separated by the a line feed character. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". If unset, the paths set in iscobol.file.prefix are used. (iscobol.binary_file_prefix is supported for backward compatibility)
iscobol.file.case	**U** / **u** = Data file names are converted to upper case. Conversion occurs before applying file prefix and file suffix. **L** / **l** = Data file names are converted to lower case. Conversion occurs before applying file prefix and file suffix. (Other) Data file names are not changed. The default value is empty, so no conversion happens.
iscobol.file.close_on_exit (boolean)	**True** = All open files are automatically closed when the program exits. **False** = Open files are left open when the program exits. The default value is False This property is overriden by the -coe compiler option. For example, if a program is compiled with -coe, files will be closed even if it sets iscobol.file.close_on_exit=false with a SET ENVIRONMENT statement before exiting. (iscobol.close_on_exit is supported for backward compatibility)
iscobol.file.connector.program	This property specifies an alternate name for the c-tree File Connector executable. The default value is "fscsc".
iscobol.file.connector.program.dcic	This property specifies an alternate name for the DCI File Connector executable. The default value is "dcic".
iscobol.file.connector.program.mfc	This property specifies an alternate name for the Micro Focus File Connector executable. The default value is "mfc".

Property	Meaning
`iscobol.file.connector.program.rmc`	This property specifies an alternate name for the RM/COBOL File Connector executable. The default value is "rmc".
`iscobol.file.connector.program.vfc`	This property specifies an alternate name for the Vision File Connector executable. The default value is "vfc".
`iscobol.file.encryption.key *`	This property specifies the encryption key that JIsam will use to deal with encrypted indexed files. See Encryption for more details. The encryption key must be 1 to 16 bytes in size and can't be spaces. The property is checked before opening the file. In Application Server and File Server environments this property should be set in the server configuration and it affects all the clients.

Property	Meaning

iscobol.file.env_naming (boolean)	**True** = The Runtime Framework searches for the name of the file among the environment variables. If found, its value is used in place of the file name:
	
	`SELECT LOGICAL FILE ASSIGN TO "PHYSICAL-FILE"`
	
	False = The Runtime Framework searches for the name of the file among the environment variables only if it is preceded by "-E":
	
	`SELECT LOGICAL-FILE ASSIGN TO "-E PHYSICAL-FILE"`
	
	The default value is False.
	
	Rules for the correct format of the property name:
	if you set the environment variable in isCOBOL properties, be sure to use lower case, as regardless of what case is used in the COBOL program, all property names must be lower-case. If you set the variable in system variables, instead, use upper case. If you set the variable via SET ENVIRONMENT, the runtime will take care of normalizing it, so either lower or upper case can be used.
	if the file name contains hyphens, replace them with underscore
	if the file name includes a path, use the path also in the property name
	Some examples:
	+------------------+--------------------------+
	| physical file name| property name |
	+------------------+--------------------------+
	| FILE1 | iscobol.file1 |
	| FILE-1 | iscobol.file_1 |
	| FILE-1.dat | iscobol.file_1.dat |
	| folder/FILE-1.dat | iscobol.folder/file_1.dat |
	+------------------+--------------------------+
	
	Alternatively, this property can be used to replace the file path in the program's SELECT statement with an environment variable. The path name must be prefaced by the '$' sign and must use the correct system's file separator.
	
	Example for Windows:
	
	`SELECT FILE1 ASSIGN TO "$STATION\FILE1"`
	
	Example for Unix:
	
	`SELECT FILE1 ASSIGN TO "$STATION/FILE1"`
	
	Note - Only one environment variable is considered when this property is set to true. If your program's SELECT statement contains a "$", the file path will be replaced with the environment variable, but the filename won't be replaced even if there is a valid variable for it.

Property	Meaning

`iscobol.file.env_naming_prefix`	When this property is used in conjunction with *iscobol.file.env_naming=True*, a prefix is stripped from the file mapping properties.
	
	For example, if you set:
	
	`iscobol.file.env_naming_prefix=dd_`
	`iscobol.file.dd_customers=customer_file`
	`iscobol.file.dd_cities=cities_file`
	
	You obtain the same effect as setting:
	
	`iscobol.file.customers=customer_file`
	`iscobol.file.cities=cities_file`
	
	(iscobol.file.env_naming.prefix is supported for backward compatibility)

`iscobol.file.errors_ok`	Configure how i-o errors are treated. A value of "0" means that if an error occurs and there are no declaratives, the program will stop. A value of "1" means that if an error occurs the program will continue. A value of "2" means that if an error occurs the program will continue only if a file-status is defined for the file. The default value is "0". (iscobol.errors_ok is supported for backward compatibility)
`iscobol.file.extend_creates (boolean)`	**True** = All files opened in EXTEND mode are treated as if they were declared OPTIONAL. If the file does not exist, it is created. **False** = Standard rules are applied. Opening non-existing files when in EXTEND mode causes an error. The default value is False. (iscobol.extend_creates is supported for backward compatibility)
`iscobol.file.extend_lock (boolean)`	**True** = OPEN EXTEND is treated as OPEN EXTEND WITH LOCK. **False** = OPEN EXTEND doesn't exclusively lock the file. The default value is True.
`iscobol.file.extra_keys_ok (boolean)`	**True** = Does not return errors when opening an indexed file that contains more keys than are described by the program. **False** = Returns a mismatch error when opening an indexed file that contains more keys than are described by the program. The default value is False.
`iscobol.file.indd`	This property specifies a custom class for the handling of files associated to iscobol.compiler.indd. The class must implement the com.iscobol.io.InddHandler interface. Refer to the javadoc installed with isCOBOL for details.

Property	Meaning
`iscobol.file.index`	This property specifies the default file system. Refer to the Indexed file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. The default value is "jisam".
`iscobol.file.index.autolock_allowed (boolean) *`	This property affects the lock behavior in the same run unit when locks are managed by the *InternalLockManager* class in a thin client or file server environment. It affects locks between a caller program and a called program as well as locks on multiple records acquired by a single program. **True =** More locks on the same record can be acquired in the same run unit. **False =** Only one lock on the same record can be acquired in the same run unit. The default value is False.
`iscobol.file.index.check_all_keys (boolean)`	**True** = check keys structure on OPEN and return error on mismatch. **False** = don't check keys structure on OPEN. The default value is False.
`iscobol.file.index.data_suffix *`	This property specifies the default extension to be applied to the data file of a JIsam or c-tree archive. The separator dot must be explicitly specified. By default, ".dat" is used.
`iscobol.file.index.`*`FileName`*	This property specifies the file system to be used with *FileName*. It overrides the setting of iscobol.file.index. *FileName* must match the physical file name declared in the COBOL program with hyphens replaced by underscores and must be lower-case regardless of the case used in the program. Here are some examples: for "F-CUST" set *iscobol.file.index.f_cust*=... for "f_cust" set *iscobol.file.index.f_cust*=... for "F_cust.dat" set *iscobol.file.index.f_cust.dat*=... **Note:** The names of the properties that are dynamically set inside the program using SET ENVIRONMENT are normalized by the runtime and therefore the above rules can be ignored. Refer to the Indexed file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. When the ASSIGN TO EXTERNAL clause is explicitly used or implied by the -cax compiler option, *FileName* must refer to the name of the environment variable and not to its value. For example, having `SELECT FILE1 ASSIGN TO EXTERNAL "ENV1"` in the source and having `ENV1=/tmp/file1` in the environment, set *iscobol.file.index.env1* not *iscobol.file.index.file1*. The default value is the one specified by iscobol.file.index.

Property	Meaning
`iscobol.file.index.index_suffix *`	This property specifies the default extension to be applied to the index file of a JIsam or c-tree archive. The separator dot must be explicitly specified. By default, ".idx" is used.
`iscobol.file.index.lock_read_anyhow (boolean) *`	**True** = The record data is returned to the program even if the record is locked. **False** = The record data is not returned to the program if a lock condition occurs. The default value is False. This property affects JIsam and c-tree file systems.
`iscobol.file.index.lock_wait (boolean) *`	**True =** Wait for the record to become unlocked if a lock condition occurs. **False** = Return the lock condition to the program. The default value is False. This feature is fully supported by c-tree RTG. It is supported also by JIsam but, in a thin client or file server environment, it works only if iscobol.file.lock_manager * is set to "com.iscobol.as.locking.InternalLockManager".

`iscobol.file.index.open_hook *`	This property specifies the name of a COBOL program that will be automatically called before opening an indexed file.
	The program receives the following parameters:
	`77 file-path pic x(300).`
	`77 file-class pic x(300).`
	The parameters hold the current pathname and file handler class.
	The COBOL program can change these parameters altering the runtime behavior.
	
	Sample programs for this feature are installed under the *sample/extend-and-customize/file-open-hook* directory of the isCOBOL SDK.

`iscobol.file.index.read_lock_test (boolean) *`	**True** = A READ WITH NO LOCK returns a lock condition if the record is locked. **False** = A READ WITH NO LOCK reads the record even if it's locked. The default value is False. This property affects JIsam and c-tree file systems.
`iscobol.file.index.strip_extension (boolean)`	**True** = The file extension is removed from the file name before opening the file. **False** = The file extension is not removed from the file name before opening the file. The default value is False. This property doesn't affect default extensions applied by the file handler, but it affects only the extension specified by the COBOL program, if any.
`iscobol.file.index.version`	This property returns the version of the file handler specified by the iscobol.file.index property.

Property	Meaning
`iscobol.file.indexed_file_prefix`	This property lists the paths in which to search for indexed data files. When a data file is opened for input or i-o, the Framework looks for the data file in every path listed by this property until it finds the data file. If the data file is not found, an error is returned. When the data file is opened for output, the Framework looks for the data file in every path listed by this property until it finds the data file, then it initializes it. If the data file is not found, then the Framework tries to create it in the first path listed by this property. Paths must be separated by the a line feed character or by the current operating system path separator. If one of the paths starts with "isf://", then paths must be separated by the a line feed character. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". (iscobol.indexed_file_prefix is supported for backward compatibility)
`iscobol.file.input`	This property specifies an alternative class with which to handle input streams (input streams are read-only sequential files). Contact your local distributor for technical specifications. For example, to let isCOBOL work with an EXTFH provider, this property can be set to: com.iscobol.extfh.ExtfhInput
`iscobol.file.input_nolock (boolean)*`	**True** = READ WITH LOCK, either explicit or implicit, doesn't lock the record if the file is open in INPUT mode. **False** = READ WITH LOCK, either explicit or implicit, locks the record even if the file is open in INPUT mode. The default value is True.
`iscobol.file.io_creates (boolean)`	**True** = All files opened in I-O mode are treated as if they were declared OPTIONAL. If the file does not exist, it is created. **False** = Standard rules are applied. Opening non-existing files in I-O mode causes an error. The default value is False. (iscobol.io_creates is supported for backward compatibility)
`iscobol.file.linesequential`	This property specifies an alternative class with which to handle line sequential files. Refer to the Line Sequential file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. The default value is "lseq8bit".

Property	Meaning
`iscobol.file.linesequential.FileName`	This property specifies the file system to be used with *FileName*. It overrides the setting of iscobol.file.linesequential. *FileName* must match the physical file name declared in the COBOL program with hyphens replaced by underscores and must be lower-case regardless of the case used in the program. Here are some examples: for "F-CUST" set iscobol.file.linesequential.f_cust=... for "f_cust" set iscobol.file.linesequential.f_cust=... for "F_cust.txt" set iscobol.file.linesequential.f_cust.txt=... Note: The names of the properties that are dynamically set inside the program using SET ENVIRONMENT are normalized by the runtime and therefore the above rules can be ignored. Refer to the Line Sequential file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. When the ASSIGN TO EXTERNAL clause is explicitly used or implied by the -cax compiler option, *FileName* must refer to the name of the environment variable and not to its value. For example, having `SELECT FILE1 ASSIGN TO EXTERNAL "ENV1"` in the source and having `ENV1=/tmp/file1` in the environment, set *iscobol.file.linesequential.env1* not *iscobol.file.linesequential.file1*. The default value is the one specified by iscobol.file.linesequential.
`iscobol.file.linesequential_N (boolean)`	**True** = Use the com.iscobol.io.DynamicLSeqMF_N class as the default for line sequential files. **False** = Use the com.iscobol.io.DynamicLSeq8bit class as the default for line sequential files. The default value is False (This property is deprecated and is provided for backward compatibility only. Use iscobol.file.linesequential=lseqmf_n instead.)

Property	Meaning
`iscobol.file.lock_manager` *	This property specifies an alternate class with which to handle locks in the Application Server (Thin Client) and File Server environments. Possible values are: **com.iscobol.as.locking.BaseLockManager** The Application Server tracks active locks on indexed files so they can be monitored by the server administration panel and by calling the A$LIST_LOCKS routine. However, the physical lock management is performed by the active file handler. **com.iscobol.as.locking.InternalLockManager** Application Server and File Server manage locks on indexed files itself without demanding the lock request to the active file handler. In Application Server environment active locks are listed by the server administration panel and can be inquired by calling the A$LIST_LOCKS routine. If this property is omitted, then locks are managed by the active file handler and are not traced by either the server administration panel nor by the A$LIST_LOCKS routine. For more information about lock managers, see Internal lock management. Note - Lock managers don't work in the Application Server if multitasking is enabled, e.g. if iscobol.as.multitasking is set to a value greater than 0 in the isCOBOL Server configuration. When multitasking is enabled, lock managers work only in the File Server.
`iscobol.file.min_rec_size`	When trailing spaces are stripped, this property specifies the minimum size of a line. The default value is 1. Trailing spaces are always stripped in print files, unless the NO CONVERSION clause is used on WRITE. Trailing spaces are stripped in disk sequential files when iscobol.file.strip_trailing_spaces (boolean) is set to *True*. (iscobol.min_rec_size is supported for backward compatibility)
`iscobol.file.open_check (boolean)` *	**True =** If a relative or sequential file is opened by a process even without using any kind of lock, no other process can lock it in exclusive mode. **False =** If a relative or sequential file is opened by a process without any locking, another process can lock it in exclusive mode. The default value is True.
`iscobol.file.outdd`	This property specifies a custom class for the handling of files associated to iscobol.compiler.outdd. The class must implement the com.iscobol.io.OutddHandler interface. Refer to the javadoc installed with isCOBOL for details.
`iscobol.file.output`	This property specifies an alternative class with which to handle output streams (output streams are write-only sequential files, like print files). Contact your local distributor for technical specifications. For example, to let isCOBOL work with an EXTFH provider, this property can be set to: com.iscobol.extfh.ExtfhOutput

Property	Meaning
`iscobol.file.output_lock (boolean)`	**True** = OPEN OUTPUT is treated as OPEN OUTPUT WITH LOCK . **False** = OPEN OUTPUT doesn't exclusively lock the file. The default value is True.
`iscobol.file.page_eject_on_close (boolean)`	**True** = Print files print a page advance record when the file is closed, unless the close contains the NO REWIND phrase. **False** = Print files don't print a page advance record when the file is closed. The default value is False. This property doesn't affect print files assigned to the spooler.
`iscobol.file.prefix`	This property lists the paths in which to search for data files. When a data file is opened for input or i-o, the Framework looks for the data file in every path listed by this property until it finds the data file. If the data file is not found, an error is returned. When the data file is opened for output, the Framework looks for the data file in every path listed by this property until it finds the data file, then it initializes it. If the data file is not found, then the Framework tries to create it in the first path listed by this property. Paths must be separated by the a line feed character or by the current operating system path separator. If one of the paths starts with "isf://", then paths must be separated by the a line feed character. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". (iscobol.file_prefix is supported for backward compatibility)
`iscobol.file.prefix_separator`	This property sets the character used by isCOBOL to define the full path name (joining the value of the *file.prefix* setting and the file name specified in the program) according to the platform.This property was introduced specifically for client-server file systems, such as c-tree. For example, if the c-tree server works on Linux while clients are on Windows, and this variable has not been set, isCOBOL inserts a backslash separator, resulting in an error. The default value is "\" for Windows and "/" for UNIX/Linux. (iscobol.file_prefix.separator is supported for backward compatibility)
`iscobol.file.relative`	This property specifies an alternative class with which to handle relative files. Refer to the Relative file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. The default value is "relative".

Property	Meaning
`iscobol.file.relative.`*`FileName`*	This property specifies the file system to be used with FileName. It overrides the setting of iscobol.file.relative. *FileName* must match the physical file name declared in the COBOL program with hyphens replaced by underscores and must be lower-case regardless of the case used in the program. Here are some examples: for "F-CUST" set iscobol.file.relative.f_cust=... for "f_cust" set iscobol.file.relative.f_cust=... for "F_cust.dat" set iscobol.file.relative.f_cust.dat=... Note: The names of the properties that are dynamically set inside the program using SET ENVIRONMENT are normalized by the runtime and therefore the above rules can be ignored. Refer to the Relative file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. When the ASSIGN TO EXTERNAL clause is explicitly used or implied by the -cax compiler option, *FileName* must refer to the name of the environment variable and not to its value. For example, having `SELECT FILE1 ASSIGN TO EXTERNAL "ENV1"` in the source and having `ENV1=/tmp/file1` in the environment, set *iscobol.file.relative.env1* not *iscobol.file.relative.file1*. The default value is the one specified by iscobol.file.relative.
`iscobol.file.relative_file_prefix`	This property lists the paths in which to search for relative data files. When a data file is opened for input or i-o, the Framework looks for the data file in every path listed by this property until it finds the data file. If the data file is not found, an error is returned. When the data file is opened for output, the Framework looks for the data file in every path listed by this property until it finds the data file, then it initializes it. If the data file is not found, then the Framework tries to create it in the first path listed by this property. Paths must be separated by the a line feed character or by the current operating system path separator. If one of the paths starts with "isf://", then paths must be separated by the a line feed character. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". If unset, the paths set in iscobol.file.prefix are used. (iscobol.relative_file_prefix is supported for backward compatibility)
`iscobol.file.remote.host *`	This property specifies the host name where the isCOBOL File Server is listening. The default value is 'localhost'.
`iscobol.file.remote.port *`	This property specifies the port where the isCOBOL File Server is listening. The default value is 10997.

Property	Meaning
`iscobol.file.remove_name_spaces (boolean)`	**True =** spaces are removed from physical file names before looking for the files on disk (e.g. "C:\t m p \ file1" is treated as "C:\tmp\file1"). **False =** spaces are not removed from physical file names before looking for the files on disk. The default value is False. **Note -** spaces are removed only from the name used by the COBOL program, not from file.prefix and current directory.
`iscobol.file.sequential`	This property specifies an alternative class with which to handle sequential files. Refer to the Binary Sequential file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. The default value for files with fixed length record is "sequential". The default value for files with variable length record is "varseq".
`iscobol.file.sequential.`*`FileName`*	This property specifies the file system to be used with *FileName*. It overrides the setting of iscobol.file.sequential. *FileName* must match the physical file name declared in the COBOL program with hyphens replaced by underscores and must be lower-case regardless of the case used in the program. Here are some examples: for “F-CUST” set iscobol.file.sequential.f_cust=... for “f_cust” set iscobol.file.sequential.f_cust=... for “F_cust.txt” set iscobol.file.sequential.f_cust.txt=... Note: The names of the properties that are dynamically set inside the program using SET ENVIRONMENT are normalized by the runtime and therefore the above rules can be ignored. Refer to the Binary Sequential file handlers table for the list of possible values. You can use either the Class-Name or the Alias (if available) to specify the file handler. When the ASSIGN TO EXTERNAL clause is explicitly used or implied by the -cax compiler option, *FileName* must refer to the name of the environment variable and not to its value. For example, having `SELECT FILE1 ASSIGN TO EXTERNAL "ENV1"` in the source and having `ENV1=/tmp/file1` in the environment, set *iscobol.file.sequential.env1* not *iscobol.file.sequential.file1*. The default value is the one specified by iscobol.file.sequential.

Property	Meaning
iscobol.file.sequential_file_prefix	This property lists the paths in which to search for sequential data files. When a data file is opened for input or i-o, the Framework looks for the data file in every path listed by this property until it finds the data file. If the data file is not found, an error is returned. When the data file is opened for output, the Framework looks for the data file in every path listed by this property until it finds the data file, then it initializes it. If the data file is not found, then the Framework tries to create it in the first path listed by this property. Paths must be separated by the a line feed character or by the current operating system path separator. If one of the paths starts with "isf://", then paths must be separated by the a line feed character. Within configuration files, the line feed character is "\n". In COBOL programs, the line feed character is x"0a". If unset, the paths set in iscobol.file.prefix are used. (iscobol.sequential_file_prefix is supported for backward compatibility)
iscobol.file.status *	This property specifies which file status codes to use. Set it to one of the following values: • com.iscobol.io.FileStatusDefault (for 2002 file status codes) • com.iscobol.io.FileStatus85 • com.iscobol.io.FileStatus74 • com.iscobol.io.FileStatusDG • com.iscobol.io.FileStatusVax • com.iscobol.io.FileStatusIBM • com.iscobol.io.FileStatusMF • com.iscobol.io.FileStatusMS For example, for RM/COBOL-85 (ANSI 85) codes set `iscobol.file.status=com.iscobol.io.FileStatus85` It's possible to create a custom set of file status codes. The class must implement the "com.iscobol.io.FileStatus" interface; contact your local distributor for technical specifications.
iscobol.file.strip_trailing_spaces (boolean)	**True =** Trailing spaces are automatically removed before writing a line sequential file. When reading, the destination item is automatically filled with spaces before the line is read. **False =** Trailing spaces are not removed before writing a line sequential file. When reading, the destination item is not cleared before the line is read. The default value is False. This property affects line sequential files on disk. In print files trailing spaces are stripped by default, unless the NO CONVERSION clause is used on WRITE. (iscobol.strip_trailing_spaces is supported for backward compatibility)

Property	Meaning
iscobol.file.suffix	This property automatically appends a suffix to all data file names. The dot that separates the file name and the file extension is automatically added by the runtime. Example: opening a file whose physical name is "arc" having *iscobol.file.suffix=dat* will open "arc.dat". (iscobol.file_suffix is supported for backward compatibility)
iscobol.file.xextfh	This property allows you to choose the file manger when EXTFH is not involved. Valid values are: **2 =** Use extended EXTFH2 32 bit **3 =** Use extended EXTFH3 64 bit
iscobol.jisam.autolock_allowed (boolean)	This property affects the lock behavior in the same run unit when using JIsam as file handler. It affects locks between a caller program and a called program as well as locks on multiple records acquired by a single program. **True =** More locks on the same record can be acquired in the same run unit. **False =** Only one lock on the same record can be acquired in the same run unit. The default value is False.
iscobol.jisam.version	Specifies the version of JIsam files created by OPEN OUTPUT. Valid values are 1 and 2. You should set this property to 1 if you plan to share the JIsam files with programs that run with an old version of isCOBOL that doesn't support JIsam version 2. The default value is 2.
iscobol.sort	This property specifies an alternative class with which to handle sort files. Contact your local distributor for technical specifications. For example, to let isCOBOL work with an external sort module, this property can be set to: com.iscobol.extfh.ExtsmSort
iscobol.sort.dir	This property specifies the path of the directory where temporary sort files are stored.
iscobol.sort.maxfiles	This property specifies the maximum amount of files used for sorting. The default value is 16.
iscobol.sort.memsize	This property specifies the maximum amount of memory used for sorting. The value is expressed in bytes. The default value is 1048576.
iscobol.sqlserver.convention	This property specifies the sign convention adopted by the COBOL program that creates the c-tree file. Possible values are: **A =** Acucobol-GT convention **D =** Data General convention **I =** IBM convention **M =** Micro Focus convention **N =** NCR COBOL convention **R =** Realia COBOL convention **V =** VAX COBOL convention The default value is A.

Property	Meaning
`iscobol.sqlserver.database`	This property specifies the name of the c-tree SQL database that the Framework must connect to. This setting is used when iscobol.sqlserver.iss is set to true.
`iscobol.sqlserver.dirlevel`	This property specifies how many parts of the file path are to be used to build the table name on c-tree SQL when iscobol.sqlserver.iss is set to true. The default value is 0.
`iscobol.sqlserver.grant`	This property specifies the permissions for the linked table. It controls how other users will be able to interact with the table. Possible values are: **0 =** no permission, access denied to other users **1 =** public permission with full access to other users **2 =** public permission with read-only access to other users The default value is 0.
`iscobol.sqlserver.iss (boolean)`	**True =** The c-tree indexed file is automatically linked into c-tree SQL during the OPEN OUTPUT. **False =** The c-tree indexed file is created as a standard c-tree file during the OPEN OUTPUT. The default value is False.
`iscobol.sqlserver.iss.mapping.`*`filename`*	This property creates a mapping between one or more physical file names and an iss file. *filename* is the name of the physical file, wildcards are supported (e.g. customers*). The value for this property is the basename (no extension) of the iss file to be used for all file names that match the pattern specified by *filename*.

Property	Meaning
iscobol.sqlserver.iss.replacement_rules	This property configures the replacements performed by the runtime on the physical file name before linking it as a table when *iscobol.sqlserver.iss* is set to true. It affects the name of the iss dictionary that will be searched by the runtime as well as the name of the table in the c-tree SQL database. By default '.' and '-' become '_'. Before this conversion takes place, you can strip '.', '-' and file extension from the file name by setting this property to the combination between one or more of the following values: **0** = don't omit any character **1** = '.' is omitted **2** = '-' is omitted **4** = the file extension (from the last '.' to the end of the name) is omitted For example, given a file whose name is MY-ARC.01.DAT, the runtime will create the table on the c-tree SQL database with the following criteria: +------------------+--------------------+ | replacement rules | table name | +------------------+--------------------+ | 0 | MY_ARC_01_DAT | | 1 | MY_ARC01DAT | | 2 | MYARC_01_DAT | | 3 | MYARC01DAT | | 4 | MY_ARC_01 | | 5 | MY_ARC01 | | 6 | MYARC_01 | | 7 | MYARC01 | +------------------+--------------------+ The property affects also the name of the ISS dictionary used by the runtime to manage the file, unless differently configured via the iscobol.sqlserver.iss.mapping.filename setting. For example, given a file whose name is MY-ARC.01.DAT, the runtime will look for the ISS dictionary with the following criteria: +------------------+--------------------+ | replacement rules | ISS dictionary | +------------------+--------------------+ | 0 | my_arc_01_dat.iss | | 1 | my_arc01dat.iss | | 2 | myarc_01_dat.iss | | 3 | myarc01dat.iss | | 4 | my_arc_01.iss | | 5 | my_arc01.iss | | 6 | myarc_01.iss | | 7 | myarc01.iss | +------------------+--------------------+ The default value is 0.
iscobol.sqlserver.isspath	This property specifies the path where iss files are located. This setting is used when iscobol.sqlserver.iss is set to true.

Property	Meaning
iscobol.sqlserver.owner	This property specifies the owner of the SQL tables. This setting is used when iscobol.sqlserver.iss is set to true.
iscobol.sqlserver.password	This property specifies the password for connecting to c-tree SQL. This setting is used when *iscobol.sqlserver.iss* is set to true in the isCOBOL configuration and COMPATIBILITY SQLIMPORT_ADMIN_PASSWORD is present in the c-tree Server configuration (ctsrvr.cfg). Note that the use of COMPATIBILITY SQLIMPORT_ADMIN_PASSWORD is discouraged for security reasons.
iscobol.sqlserver.prefix	This property specifies a prefix to be put before the name of the SQL tables. This setting is used when iscobol.sqlserver.iss is set to true.
iscobol.visionj.v_apply_signature	**True =** apply the custom comment "Created by Vision4J" in the header of new files. **False =** don't apply the custom comment "Created by Vision4J" in the header of new files. The default value is False.
iscobol.visionj.v_buffers_per_file *	This property sets the number of indexed block buffers to allocate per file. These buffers are used to improve the performance of indexed files. Each buffer is 512 bytes plus some overhead. Increasing the number of buffers can improve file performance. Decreasing the number conserves memory. The value can range from zero (no buffering) to 2097152. The default value is 64.
iscobol.visionj.v_compress_factor	This property is used to define the compression factor that is applied to Vision files. The compression factor is applied when a file is created with the WITH COMPRESSION phrase in the ASSIGN clause of the file's SELECT and the COMPRESSION CONTROL VALUE phrase is either omitted or specifies a value of 1. If the COMPRESSION CONTROL VALUE phrase specifies a value other than one, that value is used and the value of this property is ignored. This property can be set to any value within the range 0 to 100. The value 0 specifies no compression. Values from 2 to 100 are treated as a percentage that specifies how much of the space saved by file compression is removed from the compressed records. A value of 1, the default, is a special case that causes the standard default compression factor of 70 to be applied. Note that a file's compression factor is set when the file is created and cannot later be changed except by recreating or rebuilding the file.
iscobol.visionj.v_force_open	**True =** open broken files that would normally cause a "file broken" error. **False =** return a "file broken" error when opening a broken file. The default value is False.

Property	Meaning
iscobol.visionj.v_index_block_percent	This property determines the ratio of index block to data blocks that are created for a Vision file, which can help trim file size when disk space is tight. When set to a value smaller than 100, fewer index blocks are created than data blocks. When set to a value greater than 100, more index blocks are created than data blocks. The value range is 1 to 1000. The default value is 100.
iscobol.visionj.v_internal_locks	**True =** return a record or file locked condition for a record or file that was previously locked by the same run unit. **False =** don't return a record or file locked condition for a record or file that was previously locked by the same run unit. The default value is True.
iscobol.visionj.v_lock_method	This property selects which locking method Vision will use to control simultaneous access to indexed files. **0 =** Vision locks the first byte of the file for every access to the file (both reads and updates). This ensures that the process is not interfered with by another process. **1 =** Vision locks the first byte of the file for all operations except random READs or READ NEXTs. These two operations proceed without the lock. Instead, they perform some additional reads of the file, to ensure that they get consistent results. If they get inconsistent results, they are retried, this time locking the first byte as other operations do. The default value is 0.
iscobol.visionj.v_locks_per_file	This property determines the maximum number of record locks that can be held on a file by a single process. This value affects only the files that are maintaining multiple record locks. The maximum value is 32767. Setting this variable to its maximum value can waste resources and is not recommended. The default setting is 10.
iscobol.visionj.v_mark_read_corrupt	**True =** mark a file as broken if it encounters a corruption during a read or start operation. **False =** don't mark a file as broken if it encounters a corruption during a read or start operation. The default value is True.
iscobol.visionj.v_max_files *	This property sets the maximum number of files that can be opened by the current run-time session. Keeping this value small conserves memory. Many operating systems limit the number of files that can be opened by a single process, so you may have to make some adjustments there too. The maximum value is 32767. The default value is 64.

Property	Meaning
`iscobol.visionj.v_open_strict (boolean)`	**True =** receive an error status when opening a file marked as broken for INPUT. **False =** allow OPEN INPUT on broken files. The default value is False. Note - When this property is set to True, make sure you do not also have the V_FORCE_OPEN variable set to True because the settings conflict.
`iscobol.visionj.v_read_ahead (boolean) *`	**True =** turns on Vision's read-ahead logic. **False =** turns off Vision's read-ahead logic. Turning off the Vision's read-ahead logic may improve performance in cases where highly random file processing is being used. The default value is True.
`iscobol.visionj.v_seg_size`	This property sets the maximum size of a Vision file segment. Vision Version 5 allows for very large logical files by creating additional physical files (called file segments) as needed. The default value is 2 GB - 1 KB -1 (which is 2,147,482,623 or hexadecimal 0x7FFFFBFF). The default is also the maximum value allowed. For best performance, set this value as high as possible, to minimize the number of files created.
`iscobol.visionj.v_strip_dot_extension (boolean)`	**True =** strip any trailing dot extension (such as .dat) from the logical name of a data file when generating file names for index and data segments (except for the first data segment). **False =** prevent file extensions frombeing removed. The default value is True.
`iscobol.visionj.v_version`	This property specifies the version number of new Vision data files that are created by Vision4J. It accepts values from 3 to 6. Version 5 and later files are generated in a dual file format, with data records stored in one segment and overhead key information stored in another. The value 4 produces Version 4 files, which also use the dual file format, but do not support the larger maximum record size and larger block sizes available in Version 5 and later. The value 3 produces Version 3 files, in which dataand keys are stored in a single file. The default value is 6, which produces Vision files in the latest format (Version 6).

DatabaseBridge and JDBC/ESQL Configuration

(*) The asterisk after the property name means that the property is static, it’s inquired only once by the Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.esql.db2.row_data_as_bytes_threshold *`	This property is considered only when connected to a IBM DB2 database and it specifies the minimum length of CHAR and VARCHAR fields that must be managed as byte arrays instead of strings. If set to 0, all CHAR and VARCHAR fields are managed as strings. The default value is 0.
`iscobol.esql.default_param_type`	This property specifies the default type of parameters passed to stored procedures. If affects stored procedures called via CALL statement as well as stored procedures called in PL/SQL blocks via EXECUTE statement. The default parameter type specified by this property is overridden by the IN, OUT and INOUT clauses in the CALL statement as well as by storeproc.property and by the $SQL HOSTVAR directive. Possible values are: • IN • OUT • INOUT The default value is INOUT.
`iscobol.esql.error.negative (boolean)`	**True** = SQLCODE values different from 0 and 100 are returned as negative values. **False** = SQLCODE values are always positive. The default value is False. Warnings are not affected by this property. See iscobol.esql.warnings (boolean) for information about how to intercept warnings. SQLCODE values can also be changed via the iscobol.esql.sqlcode.<value>=<new-value> * property.
`iscobol.esql.indicator_trunc_on_call (boolean)`	**True** = set the indicator variable to the length of the stored procedure's output parameter value when this value doesn't fit the host variable. **False** = set the indicator variable to 0 when the stored procedure's output parameter value doesn't fit the host variable. The default value is True.

Property	Meaning

`iscobol.esql.sqlcode.<value>=<new-value> *`	This property allows you to remap SQLCODE values to custom values. For example, in order to obtain SQLCODE=1403 instead of SQLCODE=100 when no record is found, set
	
	`iscobol.esql.sqlcode.100=1403`
	
	It doesn't allow to remap SQLCODE values produced by iscobol.esql.value_sqlcode_on_no_data, iscobol.esql.value_sqlcode_on_null and iscobol.esql.value_too_many_rows as well as the value 1405 returned by programs compiled with -csqn option.
	
	This property is evaluated after iscobol.esql.error.negative (boolean). If *error.negative* is true and you wish to remap a SQLCODE value, then you need to remap the negative value.
	The following pair of settings
	
	`iscobol.esql.error.negative=false`
	`iscobol.esql.sqlcode.1847=9999`
	
	produce the same effect of
	
	`iscobol.esql.error.negative=true`
	`iscobol.esql.sqlcode.-1847=9999`

`iscobol.esql.value_sqlcode_on_no_data`	This property specifies the value of SQLCODE when a query doesn't affect any row. The following SQL statements are affected • DELETE • INSERT INTO SELECT • UPDATE The default value is 0.
`iscobol.esql.value_sqlcode_on_null`	This property specifies the value of SQLCODE when an host variable is set to null. This feature is activated by the -csqn compiler option. If the option is omitted, the program sets SQLCODE to zero when an host variable is set to null. The default value is 1405.
`iscobol.esql.value_too_many_rows`	This property specifies the value of SQLCODE when an EXEC SQL SELECT INTO statement returns more than one result. Destination host variables are however set with the values of the first result. The default value is 0.
`iscobol.esql.warnings (boolean)`	**True** = warnings are returned through SQLCA if the ESQL statement produces them. **False** = warnings are never returned through SQLCA. The default value is False.
`iscobol.jdbc.allocate_type`	This property specifies the data type of items allocated through the EXEC SQL ALLOCATE statement. It can be set to one of the constant values of java.sql.Types (https://docs.oracle.com/javase/8/docs/api/java/sql/Types.html). The default value is -16

Property	Meaning
`iscobol.jdbc.auto_connect (boolean)`	**True** = The runtime issues a CONNECT statement automatically if no connection is available for the Embedded SQL. **False** = The runtime doesn't try to automatically connect if no connection is available for the Embedded SQL. The CONNECT issued by the runtime has no parameters, it's just EXEC SQL CONNECT END-EXEC, so the necessary parameters must have been set in the configuration (e.g. iscobol.jdbc.url). The default value is False.
`iscobol.jdbc.autocommit (boolean)`	**True** = The JDBC driver is forced to automatically commit INSERT, UDATE and DELETE statements performed by the ESQL module. **False** = INSERT, UPDATE and DELETE statements are not automatically committed. It's program duty to issue a COMMIT or a ROLLBACK after them. The autocommit mode is set at the CONNECT statement. Changing this property after the connection to the database has been established has no effect. All statements issued after the last COMMIT or ROLLBACK in the runtime session will be committed or discarded on STOP RUN according to the iscobol.jdbc.on_stop_run setting. The default value is True.
`iscobol.jdbc.connection.login.timeout`	This property specifies the timeout in seconds before returning a connection refused error to the CONNECT statement. If this property is not set, the connection timeout depends on database settings.
`iscobol.jdbc.cursor.concurrency *`	Set the internal Cursor type for ESQL. Valid values are: **1007** = Cursors are read-only **1008** = Cursors are updatable **1009** = Cursors are lockable (only for MS SQL Server) Default is 1007
`iscobol.jdbc.cursor.type *`	This property determines the default cursor type used by the JDBC driver. **1** = FORWARD_ONLY allows the cursor to move forward, but not backward, through the data. **2** = SCROLL_INSENSITIVE allows the cursor to move forward and backward through the data. Changes made while the cursor is open are ignored. It provides a static view of the underlying data to which the cursor refers. **3** = SCROLL_SENSITIVE allows the cursor to move forward and backwards through the data. Changes made while the cursor is open are immediately available. It provides a dynamic view of the underlying data to which the cursor refers. The default value is 1.
`iscobol.jdbc.datasource`	This property allows a custom class to be specified for the JDBC connection. This class is used by the CONNECT statement to get the connection. The class must implement the com.iscobol.rts.MyDataSource interface. Consult the javadoc installed with isCOBOL for specifics.

Property	Meaning
`iscobol.jdbc.dateformat *`	This property specifies the format in which PIC x and PIC 9 receive the value of DATE fields from database tables. yyyy = year MM = month dd = day. All supported values are listed in the Java Documentation: https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html The default value is yyyy-MM-dd.
`iscobol.jdbc.driver`	This property specifies the name of the JDBC driver to be used by the ESQL module.
`iscobol.jdbc.driver.`*`ConnectionName`*	Please consult the DatabaseBridge documentation, chapter Working with multiple connections, for details on this property.
`iscobol.jdbc.fetch_size`	This property gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are returned at the opening of a ESQL cursor. If the value specified is 0, then the hint is ignored. The default value is 0.
`iscobol.jdbc.kept_spaces`	This property specifies the manner in which trailing spaces in ESQL host variables are handled. **-1 =** Trailing spaces are maintained. **0 =** Trailing spaces are removed. **1 =** Trailing spaces are removed and the first character of the string is always maintained, regardless of whether it is a space or not. **2 =** Trailing spaces and low-values are removed. Trailing spaces are handled before the host variable is passed to the ESQL processor. **Note -** this property affects only host variables. Constant strings (including the value of the The EDBI-WHERE-CONSTRAINT external variable) are not affected. The default value is 1.
`iscobol.jdbc.on_stop_run`	This property specifies the behavior of the JDBC driver when the run unit terminates. **commit =** Executes a COMMIT statement, then a DISCONNECT statement. **rollback =** Executes a ROLLBACK statement, then a DISCONNECT statement. Note that the ROLLBACK will have effect only if you're not working in autocommit mode, e.g. iscobol.jdbc.autocommit (boolean) must be set to false. **none =** Executes only a DISCONNECT statement. A COMMIT or ROLLBACK is performed depending on the database defaults. The default value is none.

Property	Meaning
`iscobol.jdbc.options`	This property sets a list of options to be passed to the JDBC driver. Every option consists of a name and a value, separated by the equals (=) symbol. Options are separated by commas. For example: `iscobol.jdbc.options=Option1=Value1,Option2=Value2` Options must be set before connecting. **Note -** Due to JDBC rules, options are evaluated only if user and password are not specified in the CONNECT statement. If you wish to specify user and password in the CONNECT statement, then you should rely on a custom CONNECT implementation in order to set JDBC options. See iscobol.jdbc.datasource for details.
`iscobol.jdbc.password`	This property specifies the password to log in the database. You should set this property only if the password is not specified neither in iscobol.jdbc.url nor by the CONNECT statement.
`iscobol.jdbc.thread_connection (boolean)`	**True =** Each thread created by PERFORM THREAD or CALL THREAD uses a separate database connection. The thread code is responsible to create the connection. When working with the DatabaseBridge, EDBI subroutines take care of managing the multiple thread connections. **False =** All threads created by PERFORM THREAD or CALL THREAD share the same database connection. The default value is False.
`iscobol.jdbc.timeformat *`	This property specifies the format in which PIC X and PIC 9 receive the value of TIME fields from database tables. All supported values are listed in the Java Documentation: https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html Default value is HH:mm:ss.SSS.
`iscobol.jdbc.timestampformat *`	This property specifies the format in which PIC X and PIC 9 receive the value of TIMESTAMP fields from database tables. All supported values are listed in the Java Documentation: https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html Default value is yyyy-MM-dd HH:mm:ss.SSS.

`iscobol.jdbc.url`	This property specifies the prefix of the JDBC URL used by the ESQL module. Possible values depend on the JDBC driver being used.
	
	The URL could include user and password, or you can specify user and password separately by setting iscobol.jdbc.user and iscobol.jdbc.password.
	
	For example, the following setting:
	`iscobol.jdbc.url=jdbc:oracle:thin:scott/tiger@myhost:1521:orcl`
	is equivalent to:
	`iscobol.jdbc.url=jdbc:oracle:thin:@myhost:1521:orcl`
	`iscobol.jdbc.user=scott`
	`iscobol.jdbc.password=tiger`

Property	Meaning
`iscobol.jdbc.url.`*`ConnectionName`*	Please consult the DatabaseBridge documentation, chapter Working with multiple connections, for details on this property.
`iscobol.jdbc.user`	This property specifies the user to log in the database. You should set this property only if the user is not specified neither in iscobol.jdbc.url nor by the CONNECT statement.

Property	Meaning
`iscobol.easydb.commit_count`	Specifies the COMMIT COUNT for EDBI routines. When this property is set to a value greater than 0, a COMMIT is issued after this number of operations. WRITE, REWRITE, and DELETE are counted; READ, START, and READ NEXT are not. The default value is 0. iscobol.jdbc.autocommit (boolean) must be set to false in the configuration, otherwise all the operations are automatically committed. **Note -** routines generated by the EDBIIS command consider this setting only if the -cc option was used.
`iscobol.easydb.commit_count.`*`ConnectionName`*	Specifies the COMMIT COUNT for EDBI routines. It affects only the connection identified by *ConnectionName*. When this property is set to a value greater than 0, a COMMIT is issued after this number of operations. WRITE, REWRITE, and DELETE are counted; READ, START, and READ NEXT are not. The default value is 0. iscobol.jdbc.autocommit (boolean) must be set to false in the configuration, otherwise all the operations are automatically committed. **Note -** routines generated by the EDBIIS command consider this setting only if the -cc option was used.
`iscobol.easydb.connection_name.`*`FileName`*	Please consult DatabaseBridge documentation, chapter Working with multiple connections, for details on this property.
`iscobol.easydb.date_cutoff`	This property uses a two-digit value and establishes the two-digit year that will be interpreted by the program as being in the 20th Century and the two-digit year that will be interpreted by the program as being in the 21st Century. For example, consider setting: `iscobol.easydb.date_cutoff=30` In this case, 2000 will be added to the two-digit year that are smaller than "30" (or whatever value you give to this variable), and will therefore make them part of the 21st Century. 1900 will be added to the two-digit year that are larger than "30", making them part of the 20th Century. A COBOL date like 99/10/10 will be translated into 1999/10/10. A COBOL date like 00/02/12 will be translated into 2000/02/12. The default value is 20. (iscobol.easydb_date_cutoff is supported for backward compatibility)

Property	Meaning
`iscobol.easydb.dirlevel`	This property gives you a method of mapping filenames with directories to the EDBI routine. It determines how many levels of the directory to use as part of the table name. If you set this property to 0, it means no level behavior (default). If you set it to a positive value, it determines how many directory names to keep. For example assuming that a COBOL program opens a file like "/usr/temp/users/invoice", if *easydb.dirlevel*=0, the EDBI routine will reference a table named "invoice"; if *easydb.dirlevel*=1, the EDBI routine will reference a table named "usersinvoice"; if *easydb.dirlevel*=2, the EDBI routine will reference a table named "tempusersinvoice"; and so on... When this property is set to a value greater than zero, it generates a mismatch between the name of the table and the name of the EDBI routine, so it's necessary to set iscobol.easydb.mapping as well. With reference to the above example, you should set *iscobol.easydb.mapping=*invoice=invoice*. The default value is 0.
`iscobol.easydb.inv_date`	This property is used to establish an invalid date (such as 2000/02/31) in order to avoid problems that can occur when an incorrect date format has been written to the database. The date must be specified as a eight digits string that is the concatenation between year, month and day. For example, to use 1st January 2000 as valid date to replace invalid dates, set: `iscobol.easydb.inv_date=20000101` The default value is 99991230. During record insertion, if the COBOL field contains an invalid date, the date specified by this property is written to the database field. After record read, if the date value matches with the value of this property, the COBOL field is set to zero. (iscobol.easydb_inv_date is supported for backward compatibility)
`iscobol.easydb.julian_base_date`	This property specifies the base date to resolve julian dates. The value must be specified in the format YYYYMMDD. For example, to use the 1st January 2000 as base date to resolve julian dates, set: `iscobol.easydb.julian_base_date=20000101` The default value is 16000101. This property can be set to dates that are greater than the 1st January 1600. If you need a lower date as base date (E.g. 00010101) then you can't rely on the automatic handling of Julian dates, but you need to write your own date conversion routines. See iscobol.compiler.easydb.julian_routines=<cbdb>;<dbcb> for more information. (iscobol.easydb_julian_base_date is supported for backward compatibility)

Property	Meaning
`iscobol.easydb.limit_dropdown`	This property limits the number of queries performed for a START operation. . Possible values are: **0** - Perform all the queries in the drop down. **1** - Only for START with SIZE clause, stop after all records matching the START columns have been exhausted. **2** - Only for START without SIZE clause, stop after all records matching the START columns have been exhausted. **3** - Regardless of the kind of START, stop after all records matching the START columns have been exhausted. The default value is 0. See iscobol.easydb.limit_dropdown for more information.

Property	Meaning

iscobol.easydb.mapping	This property allows you to associate certain filenames with particular EDBI routines. This enables you to use one EDBI routine for many different files with the same data structure. Multiple mappings must be separated by space. For example, setting:
	
	`iscobol.easydb.mapping=invoice2008=invoice invoice2009=invoice`
	
	or the equivalent
	
	`iscobol.easydb.mapping=invoice2008=invoice \`
	`invoice2009=invoice`
	
	means that the Dynamic Filesystem Interface will redirect I/O done from any COBOL program that uses "invoice2008" and "invoice2009" as a physical file basename to EDBI-invoice.
	You can also use the wildcard "*" character to simplify file name associations, where "*" matches any number of characters. Note that "*" is the only wildcard supported.
	For example,
	
	`iscobol.easydb.mapping=invoice*=invoice`
	
	will match all of the above.
	The parsing of iscobol.easydb.mapping value is interrupted at the first match found. You should pay attention to the order mappings are listed in order to avoid an unexpected mapping within similar file names. For example, suppose that, in addition to the invoice files shown above you have a different file named "invoices" and you want it to be managed by the routine EDBI-file1. The following setting is wrong:
	
	`iscobol.easydb.mapping=invoice*=invoice \`
	`invoices=file1`
	
	Because "invoice*" would be valid also for the file name "invoices" and so that file would be associated to the EDBI-invoice routine as well. The correct setting in this case is:
	
	`iscobol.easydb.mapping=invoices=file1 \`
	`invoice*=invoice`
	
	Dots in the file name are translated to underscores by the DatabaseBridge. If you need to map a file whose name has an extension, you need to be aware of this rule. For example, if you want that "file1.db" is managed by EDBI-file1, set:
	
	`iscobol.easydb.mapping=file1_db=file1`
	
	If no matches are found in *iscobol.easydb.mapping*, the Dynamic Filesystem Interface will look for matches in the deprecated *iscobol.filename.mapping* configuration property. This property is still supported for backward compatibility.

Property	Meaning
`iscobol.easydb.max_date`	This property is used to establish a high-value date in order to avoid problems in cases where invalid dates have been incorrectly written to the database. The date must be specified as a eight digits string that is the concatenation between year, month and day. For example, to use 31th December 2099 as valid date to replace dates set to high-values, set: `iscobol.easydb.max_date=20991231` The default value is 99991231. During record insertion, if the COBOL field contains high-value, the date specified by this property is written to the database field. After record read, if the date value matches with the value of this property, the COBOL field is set to high-value. (iscobol.easydb_max_date is supported for backward compatibility)
`iscobol.easydb.min_date`	This property is used to establish a low-value (0 or space) date in order to avoid problems that can occur when invalid dates have been incorrectly written to the database. The date must be specified as a eight digits string that is the concatenation between year, month and day. For example, tuse 1st January 2000 as valid date to replace dates set to low-values, set: `iscobol.easydb.min_date=20000101` The default value is database dependent. During record insertion, if the COBOL field contains low-value, zero or spaces, the date specified by this property is written to the database field. After record read, if the date value matches with the value of this property, the COBOL field is set to zero. (iscobol.easydb_min_date is supported for backward compatibility)
`iscobol.easydb.mysql_row_limit`	This property specifies the weight of the cursors used to read data from MySQL. By default the MySQL JDBC driver loads in memory all the records of a table when the COBOL program performs a Start. This behavior might consume resources and affect performance when working on huge tables. If iscobol.compiler.easydb.light_cursors was set to a value grater than 0 at compile time, then EDBI subroutines read data using a paging logic in order to work around the driver behavior. The default value is 100. **Note -** routines generated by the EDBIIS command consider this setting only if the -dmld or -dmlu options were used.

Property	Meaning
`iscobol.easydb.postgres_row_limit`	This property specifies the weight of the cursors used to read data from PostgreSQL. By default the PostgreSQL JDBC driver loads in memory all the records of a table when the COBOL program performs a Start. This behavior might consume resources and affect performance when working on huge tables. If iscobol.compiler.easydb.light_cursors was set to a value grater than 0 at compile time, then EDBI subroutines read data using a paging logic in order to work around the driver behavior. The default value is 100. **Note -** routines generated by the EDBIIS command consider this setting only if the -dpld or -dplu options were used.
`iscobol.easydb.prefix`	This property specifies the prefix that the runtime should put before the EDBI routine name before calling it. For example, setting this property to "ora", when the program opens FILE1, the runtime will call ORAEDBI_FILE1. By default, no prefix is used.
`iscobol.easydb.prefix.`*`ConnectionName`*	Please consult the DatabaseBridge documentation, chapter Working with multiple connections, for details on this property.

Property	Meaning
iscobol.easydb.replacement_rules	This property configures the replacements performed by the EDBI routines on the physical file basename before using it as database table name. By default '.' and '-' become '_'. Before this conversion takes place, you can strip '.', '-' and file extension from the file name by setting this property to the combination between one or more of the following values: **0** = don't omit any character **1** = '.' is omitted **2** = '-' is omitted **4** = the file extension (from the last '.' to the end of the name) is omitted For example, given a file whose name is MY-ARC.01.DAT, the runtime will look for the table on the database with the following criteria: +------------------+--------------------+ | replacement rules | table name | +------------------+--------------------+ | 0 | MY_ARC_01_DAT | | 1 | MY_ARC01DAT | | 2 | MYARC_01_DAT | | 3 | MYARC01DAT | | 4 | MY_ARC_01 | | 5 | MY_ARC01 | | 6 | MYARC_01 | | 7 | MYARC01 | +------------------+--------------------+ The property affects also the name of the EDBI routine used by the runtime to manage the file, unless differently configured via the iscobol.easydb.mapping setting. For example, given a file whose name is MY-ARC.01.DAT, the runtime will look for the EDBI routine with the following criteria: +------------------+--------------------+ | replacement_rules | EDBI routine | +------------------+--------------------+ | 0 | EDBI_MY_ARC_01_DAT | | 1 | EDBI_MY_ARC01DAT | | 2 | EDBI_MYARC_01_DAT | | 3 | EDBI_MYARC01DAT | | 4 | EDBI_MY_ARC_01 | | 5 | EDBI_MY_ARC01 | | 6 | EDBI_MYARC_01 | | 7 | EDBI_MYARC01 | +------------------+--------------------+ The default value is 0.

Property	Meaning
`iscobol.easydb.start_on_specific_table (boolean)`	**True** = When a START is performed on a multi-record FD, only the table related to the current record type is used. **False** = When a START is performed on a multi-record FD, all the tables related to the FD are used. The default value is False. **Note -** routines generated by the EDBIIS command consider this setting only if the -esst option was used.
`iscobol.easydb.wait_for_lock (boolean)`	**True** = EDBI routines wait for the lock to be released. The EDBI routine must have been generated with *easydb.oracle.wait_for_locks=1*. **False** = EDBI routines return the lock condition. The default value is False. This feature is currently provided only for Oracle. **Note -** routines generated by the EDBIIS command consider this setting only if the -owfl option was used.
`iscobol.edbi.notnum.tracefile`	This property specifies the name of the trace file generated by EDBI subroutines for cases of *not numeric data in numeric field*. The default value is TRACENUM. **Note -** routines generated by the EDBIIS command consider this setting only if the -t option was used.

isCOBOL Server (thin client) Configuration

isCOBOL Server properties listed below cannot be set by SET ENVIRONMENT within the program. They must appear in the configuration passed to the isCOBOL Server at startup.

(*) The asterisk after the property name means that the property is read every time a Client connects. Other properties instead are read only at server startup.

Property	Meaning
`iscobol.as.alias.AliasName *`	This property defines an alias. Aliases are evaluated when iscobol.as.use_aliases (boolean) is set to true. For more information see Working with Aliases.
`iscobol.as.appserver (boolean)`	**True** = Start the Class Server along with isCOBOL Server. **False** = Don't start the Class Server along with isCOBOL Server. This property is evaluated only if *iscobol.as.fileserver* is set to *true*, otherwise the Class Server is always started. The default value is False.

Property	Meaning
`iscobol.as.authentication *`	This property defines how the users are authenticated in Application Server environment. **0** = No password required. **1** = A password is required only for admin functions. **2** = A password is always required. The default value is 1.

`iscobol.as.check_alive_interval`	This property activates a "check alive" communication between the isCOBOL Server and the connected Clients in Thin Client environment. If the communication fails, then the Server sends a kill signal to the Client. The response of the Client to the kill signal may be affected by the iscobol.as.stop_thread * setting.
	
	The property must be set to two numeric values separated by space. The first value specifies the interval in seconds between a ping and another. The second value is the response timeout in seconds.
	Optionally, you can specify other two numeric values that specify how many times the isCOBOL Server should retry to communicate with a Client before considering it dead and the delay in seconds between these retries.
	
	For example, in order to perform a check each 5 minutes with a timeout of a minute, set:
	
	`iscobol.as.check_alive_interval=300 60`
	
	In order to retry up to 5 times in 5 minutes before considering a Client dead, set:
	
	`iscobol.as.check_alive_interval=300 60 5 60`
	
	In order to have a trace of the checks that failed, set iscobol.as.logging (boolean) and iscobol.as.logging.exception (boolean) to true.
	
	By default the check alive feature is disabled.

`iscobol.as.clientupdate.propfile`	This property specifies the name of a custom *swupdater.properties* file to be used instead of the default *swupdater.properties*. While the default *swupdater.properties* includes just the *iscobol* and *iscobolNative* packages, this custom file can contain any package. This is useful to instruct the client to download additional items (e.g. custom items like programs that will be called via CALL CLIENT) in addition to the isCOBOL libraries.
`iscobol.as.clientupdate.site`	This property specifies the URL of an HTTP server where Clients can connect to look for updates. The HTTP server could be: • a external HTTP Server like IIS or Apache • a separate isCOBOL Server started with -hs option • the current isCOBOL Server started with both -as and -hs options The URL is in the form *http://servername:port*. The isUpdater utility, automatically invoked by the isCOBOL Client, will append "swupdater.properties" to the URL in order to download that configuration file. For example, setting *iscobol.as.clientupdate.site=http://192.168.0.1:10996* will cause isUpdater to download the file "http://192.168.0.1:10996/swupdater.properties" via HTTP GET method.

Property	Meaning
`iscobol.as.clientupdate.version`	This property specifies the isCOBOL build required client side. If the Client is running a build whose number is less than the value of this property, then the Client will automatically update itself before starting any COBOL program. By default, the value of this property matches with the build of the runtime library installed in the isCOBOL Server.
`iscobol.as.debugport_range`	This property specifies the range of ports that can be used to connect remote debuggers. The property is evaluated only when iscobol.as.multitasking is set either to 1 or 2. The debugger will use the first available port in this range, unless a specific port was specified through the -debugport option on the Client's command line. The value can be expressed in two ways: • a list of port numbers separated by comma, e.g. "9991,9992,9993.9994", • the minimum port number and the maximum port number separated by hyphen, e.g. "9991-9994". The default value is 9999-10099.
`iscobol.as.digest`	This property specifies the algorithm used by isCOBOL Server to encrypt passwords. The same algorithm is used by the A$GET_DIGEST library routine. Possible values are: **LEGACY** = Use the same hash method used by isCOBOL 2012 R2 and previous releases **MD5** = Use MD5 **SHA-1** = Use SHA-1 The file *password.properties* generated with a specific algorithm is not compatible with an isCOBOL Server that uses a different algorithm. The default value is SHA-1.
`iscobol.as.fileserver (boolean)`	**True** = Start File Server services along with isCOBOL Server . **False** = Don't start File Server services along with isCOBOL Server . The default value is False.
`iscobol.as.fileserver.port`	This property specifies the port used by the File Server. The default value is 10997.
`iscobol.as.hook`	This property specifies the name of the hook program that is be automatically executed by the Application Server each time a client connects. See Hook program for more information.
`iscobol.as.httpserver (boolean)`	**True** = Start the HTTP Server feature. **False** = Don't start the HTTP Server feature. The default value is False.
`iscobol.as.httpserver.port`	This property specifies the port used by the HTTP Server. The default value is 10996.

Property	Meaning
iscobol.as.httpserver.root	This property specifies the base directory of the HTTP Server. By default, the isCOBOL Server working directory is used.
iscobol.as.ide (boolean)	**True** = Allow remote IDEs to compile and run programs. **False** = Don't serve remote IDEs. The default value is False.
iscobol.as.info.arguments	This property returns the arguments of the client command-line.
iscobol.as.info.entering	This property returns the client status to the hook program. There are two possible values: **1** = Program starting **0** = Program exiting
iscobol.as.info.host	This property returns the hostname of the current client.
iscobol.as.info.program	This property returns the program name of the client command-line.
iscobol.as.info.userid	This property returns the user ID used by the client to connect.
iscobol.as.info.username	This property returns the user name used by the client to connect.
iscobol.as.logfile	This property specifies the path of the log file for the Application Server activities.
iscobol.as.logging (boolean)	**True** = Application Server activities are traced and logged. **False** = Tracing is disabled. The default value is False.
iscobol.as.logging.exception (boolean)	**True** = Print exceptions caused by remote method invocation (RMI). **False** = Hide exceptions caused by remote method invocation (RMI). The default value is False. This property is evaluated only when iscobol.as.logging (boolean) is set to true.
iscobol.as.max_connections	This property sets the maximum number of concurrent connections allowed by the isCOBOL Server. Valid range: 1 ~ 2147483647. The default value is 512.

Property	Meaning
`iscobol.as.multitasking`	This property specifies if the isCOBOL Server should create a new thread in the current JVM or a new separate process when a client connects. Possible values are: **0** = Create a new thread for every client. **1** = Create a separate process for every client. **2** = Create a new thread for clients launched without -d option and create a separate process for clients launched with -d option. Programs loaded from iscobol.remote.code_prefix, File Server clients, isCOBOL utilities and the isCOBOL Server Administration Panel are not affected by this property, they always run as internal threads. Setting *iscobol.as.multitasking=2* is useful for production environments if you need to debug without blocking other users. Multiple clients launched with -d option must use different debug ports (see -debugport option in Format 6 of isCOBOL Client usage). When *iscobol.as.multitasking=0*, if a client starts in debug mode, all the other clients can't work until the debug session terminates. Clients running in a separate task due to this property are isolated from other clients and therefore • they can't get the list of connected users by calling A$LIST_USERS, • they can't get the list of active locks by calling A$LIST_LOCKS, • they can't share customer information by calling A$USERINFO, • they can't send messages to other clients by calling A$SEND_MESSAGE. Note - Lock managers activated by iscobol.file.lock_manager * don't work in the Application Server if multitasking is enabled. When multitasking is enabled, lock managers work only in the File Server. The default value is 2.
`iscobol.as.panel.refresh_timeout *`	This property sets the timeout in seconds for the automatic refresh of the isCOBOL Server Administration Panel lists. The automatic refresh must be activated by the user by clicking on the "Auto refresh" check box in the tool-bar. The value -1 disables the automatic refresh feature. The default value is 30.
`iscobol.as.password_file`	This property specifies an alternate name and path for the *password.properties* file. Example: `iscobol.as.password_file=/etc/as_pwd.txt` The default value is "password.properties".

Property	Meaning
`iscobol.as.stop_thread *`	This property affects the killing of Client connections performed by the isCOBOL Server. A connection may be killed in three conditions: • if the Client is terminated via the command iscclient -kill. • if the Client is terminated via the isCOBOL Server's administration panel • if the Client is terminated by the "check alive" feature of the isCOBOL Server (see iscobol.as.check_alive_interval) When set to a value of zero or greater it specifies the number of seconds to wait for the Client to terminate by cleaning up anything it needs to do before dying. When the time expires, if the Client is still alive, then it is terminated regardless of its status. When set to -1, no timeout is used and the Server waits for the Client to terminate in a clean way, even if it means to wait endlessly. The default value is -1.
`iscobol.as.use_aliases (boolean)`	**True** = The name of the program specified in the Client command line is searched among aliases defined in the server configuration file. **False** = The name of the program specified in the Client command line is actually the program to start. The default value is False. For more information see Working with Aliases.
`iscobol.net.ssl.key_store`	If this properties contains a path, then isCOBOL Server and LoadBalancer will consider that path as a JKS keystore containing a certificate and it will accept only SSL connections: the certificate must contain private and public key and must be suitable for a server. See TLS/SSL support for details.
`iscobol.net.ssl.key_store_password`	This property specifies the password of the keystore when requested. See TLS/SSL support for details.

The following properties are considered by both isCOBOL Server and Client

`iscobol.hostname`	This property specifies the host name of the machine that is running the server in an Application Server environment. In the Client configuration it is possible to specify multiple values separated by comma. It's good practice to have the same number of values in iscobol.port. The client will attempt to connect to the fist available hostname and port pair. Hostnames and ports are paired from the first in the list to the last, such as hostname1:port1, hostname2:port2 and so on. If the numbers of specified hostnames and ports do not match, the last in the shorter list will be used for creating all remaining pairs. The default value is 127.0.0.1.

iscobol.port	This property specifies the port used by the Application Server. In the Client configuration it is possible to specify multiple values separated by comma. It's good practice to have the same number of values in iscobol.hostname. The client will attempt to connect to the fist available hostname and port pair. Hostnames and ports are paired from the first in the list to the last, such as hostname1:port1, hostname2:port2 and so on. If the numbers of specified hostnames and ports do not match, the last in the shorter list will be used for creating all remaining pairs. The default value is 10999.
iscobol.runtime.cs.version	This property returns the version number of the client/server.

The following properties are to be used client side only:

iscobol.net.ssl.trust_store	If this property contains a path, then isCOBOL Client and the HTTPClient class will use an SSL connection, getting information about the sever certificate from the JKS keystore indicated by the path: a special value of '*' directs the framework to use the system Java keystore. See TLS/SSL support for details.
iscobol.net.ssl.trust_store_password	This property specifies the password of the keystore when requested. See TLS/SSL support for details.
iscobol.remote_conf	This property specifies a remote configuration file to be used in an Application Server environment. The settings of this configuration file are appended to the settings in the isCOBOL Server's configuration, if any, and the resulting configuration is used in the runtime session running within the isCOBOL Server. This property affects both the isCOBOL Client in a thin client environment and the isCOBOL Framework calling remote programs. This property must be set client side but it must point to a file located in the server machine where isCOBOL Server is running.
iscobol.user.name	This property allows you to set the user name to log in to the isCOBOL Server's services: Application Server, File Server and Remote Calls. The property also specifies the value returned in the USER-ID data item of the SYSTEM-INFORMATION group item.
iscobol.user.password	This property allows you to set the user password to log in to the isCOBOL Server's services: Application Server, File Server and Remote Calls. (iscobol.user.passwd is supported for backward compatibility)

LoadBalancer Configuration

LoadBalancer properties can be set exclusively in the configuration file passed as parameter on the iscblancer command-line.

Property	Meaning
`iscobol.balancer.hostname`	This property specifies the host name of the machine that is running the LoadBalancer. The default value is 127.0.0.1.
`iscobol.balancer.logfile`	This property specifies the path of the log file for the LoadBalancer activities.
`iscobol.balancer.logging (boolean)`	**True** = LoadBalancer activities are traced and logged. **False** = Tracing is disabled. The default value is False.
`iscobol.balancer.port`	This property specifies the port used by the LoadBalancer. The default value is 10999.
`iscobol.balancer.update.interval`	This property specifies the number of seconds between the check-alive operations performed by the LoadBalancer. The default value is 60.
`iscobol.balancer.update.timeout`	This property specifies the connection timeout for the check-alive operations performed by the LoadBalancer. The default value is 60.

Print Configuration

Property	Meaning
`iscobol.print.default_font`	This property specifies the default font to be used while printing text for which no font was specified. The isCOBOL Framework allows you to specify the font for the whole print job, for specific text or for page columns via the WIN$PRINTER routine and the P$ routines. When these routines are not involved, an undefined font is used. This property allows you to specify which font should be used instead in these cases. The value format is: FontName-FontStyle-FontDim *FontName* is the name of the font, i.e. Consolas. *FontStyle* is the style of the font such as bold, italic or bolditalic. *FontDim* is the dimension of the font. If *FontName* cannot be found, then *FontStyle* and *FontDim* are applied to the undefined font loaded by the runtime.
`iscobol.print.memory (boolean)`	**True =** Print jobs are stored in memory. As a result, the process is faster but more memory is occupied. **False =** Print jobs are stored on the hard disk. One or more temporary files will be created in the user's Temp directory. In a thin client environment this property must be set on the machine where the print job is generated, that by default is the client machine. If you moved the print job to the server side via WINPRINT-SET-PRINTER-AS, then set this property in the server side configuration. The default value is False.
`iscobol.print.pdf_async (boolean)`	**True =** The close of the print file is performed in a separate thread, allowing the program to continue with the rest of the operations. The STOP RUN statement doesn't terminate the run unit until the print thread is terminated. **False =** The close of the print file is performed in the current thread, so the program waits before continuing with the rest of the operations. This setting affects print files assigned to "-P PDF". Setting this property to true is not suggested if the program needs to automatically open the PDF file when the print job is completed. The default value is False.
`iscobol.print.preview.icon`	This property specifies a custom icon for the print preview dialog shown when you close a file whose physical name is "-P PREVIEW". It must point to a file of type BMP, JPG, GIF, ICO and PNG.
`iscobol.print.preview.title`	This property specifies the title of the print preview dialog shown when you close a file whose physical name is "-P PREVIEW". The default title is "Print Preview".

Property	Meaning
`iscobol.print.spooler_async (boolean)`	**True =** The close of the print file is performed in a separate thread, allowing the program to continue with the rest of the operations. The STOP RUN statement doesn't terminate the run unit until the print thread is terminated. **False =** The close of the print file is performed in the current thread, so the program waits before continuing with the rest of the operations. This setting affects print files assigned to the system spooler (e.g. print files assigned to "-P SPOOLER"). The default value is True.
`iscobol.printer.channels`	This configuration variable is used to define and print to printer channels C01-C12. Specify the line numbers for each channel. Null entries are ignored. Those channels that have line number zero, function-names S01-S052, CSP, or are undefined, are set to line 1. You can specify only a single line number for each channel. Example: `iscobol.printer.channels=1:3::3` In this example C01 equals 1, C02 and C04 equal 3, while C03 equals 1 because it's undefined. If a print statement specifies channel C03, the line is printed at line 1. Any WRITE BEFORE/AFTER PAGE statements cause positioning to be at line 1. Each line advance increases the line number by one. A request to skip to a line number less than or equal to the current line causes a new page to begin. The appropriate number of line feeds are then generated.
`iscobol.printer.dialog.always (boolean)`	**True =** The *Choose Printer* dialog is shown each time the program opens a print file on a pipe. If the user doesn't choose a printer, then the OPEN fails. **False =** The *Choose Printer* dialog is never automatically shown by the runtime. The default value is False.

The following properties specify the default attributes for PDF prints. They affect PDF files created by writing to a file assigned to "-P PDF" or by using the *Save As* function in the print preview dialog.

Note that these values are read on the side where the printing takes place, e.g. on the client side if the PDF is written by the client, on the server side otherwise.

The print attributes can be changed at runtime by calling the WINPRINT-SET-ATTRIBUTE function.

Property	Meaning
`iscobol.print.attribute.author`	This property specifies the author of the PDF document. It can be any text.

Property	Meaning

iscobol.print.attribute.encryption	This property allows you to activate encryption. It takes a numeric bitwise value where each bit sets a specific feature. If this value is set to 0 then no encryption takes place.
	
	You can use the following constants, defined in isprint.def, to activate the desired feature:
	
	`78 pdfcrypt-no value 0.`
	`78 pdfcrypt-std-40 value 1.`
	`78 pdfcrypt-std-128 value 2.`
	`78 pdfcrypt-aes-128 value 3.`
	`78 pdfcrypt-no-metadata value x#08.`
	`78 pdfcrypt-embedded-files-only value x#10.`
	
	`78 pdfcrypt-allow-printing value x#0100.`
	`78 pdfcrypt-allow-modify-content value x#0200.`
	`78 pdfcrypt-allow-copy value x#0400.`
	`78 pdfcrypt-allow-modify-annotations value x#0800.`
	`78 pdfcrypt-allow-fill-in value x#1000.`
	`78 pdfcrypt-allow-screenreaders value x#2000.`
	`78 pdfcrypt-allow-assembly value x#4000.`
	`78 pdfcrypt-allow-degraded-printing value x#8000.`
	`78 pdfcrypt-all-permissions value x#FF00.`
	
	Permissions are applied only if combined with a valid encryption, otherwise *all-permissions* is assumed.
	
	Usage example:
	
	`iscobol.print.attribute.encryption=258`
	
	The resulting PDF will be printable, but it will not be possible to add annotations or copy the text to clipboard, as 258 is the sum between pdfcrypt-std-128 (2) and pdf-crypt-allow-printing (256).
	
	The default value is 0.

iscobol.print.attribute.expires	This property specifies the custom property "Expires". It can be any text.
iscobol.print.attribute.font_default	This property affects the generation of PDF files (e.g. files whose physical name starts with "-P PDF")and it specifies the name of the font to be used if the fonts set in the print job cannot be found in the system. Usage example: *iscobol.print.attribute.font_default=arial*
iscobol.print.attribute.font_folder	This property specifies the folders where the fonts used in the PDF document are installed. You can specify multiple folders separated by pipe, e.g. "C:\\myCustomFonts|C:\\WINDOWS\\Fonts". The fonts loaded from these folders are just referenced in the PDF. See Font embedding in PDF files for more information.

Property	Meaning
`iscobol.print.attribute.font_folder_embed`	This property specifies the folders where the fonts used in the PDF document are installed. You can specify multiple folders separated by pipe, e.g. "C:\\myCustomFonts|C:\\WINDOWS\\Fonts". The fonts loaded from these folders are physically embedded in the PDF. See Font embedding in PDF files for more information.
`iscobol.print.attribute.jpeg`	This property specifies the compression applied to images in the PDF document. It can be 0 if you want to keep images unchanged or it can range from 1 to 100 to indicate the image quality, where 1 is the lowest quality and 100 is the highest quality. When this attribute is set, all images are internally translated to jpeg; this will remove transparency, if any. The default value is 0.
`iscobol.print.attribute.keywords`	This property specifies the keywords of the PDF document. It can be any text.
`iscobol.print.attribute.owner_password`	This property specifies the password of the owner of the document. If this value is not set, then a random password is created. It works only along with iscobol.print.attribute.encryption.
`iscobol.print.attribute.pdfa`	This property allows you to create a PDF/A document following a specific standard. Possible values are "PDF/A-1A" and "PDF/A-1B", case insensitive. This property must be set in conjunction with either iscobol.print.attribute.font_folder or iscobol.print.attribute.font_folder_embed as all the fonts must be available. If this property is not set, then a standard PDF is created.
`iscobol.print.attribute.subject`	This property specifies the subject of the PDF document, it can be any text.
`iscobol.print.attribute.title`	This property specifies the title of the PDF document, it can be any text.
`iscobol.print.attribute.user_password`	This property specifies the password of the user of the document. If this value is not set, then a default password is used as specified in the PDF specifics. It works only along with iscobol.print.attribute.encryption.

IDE Reports Export feature Configuration

Export to Excel feature

The following properties are evaluated only when a Report designed with the IDE is printed to XLS or XLSX file as described in Exporting to Excel file. They're not considered by the Grid and List-Box's export features.

Property	Meaning
`iscobol.export.excel.cell_ignore_background (boolean)`	**True =** Ignores the background color of the cell. **False =** Replicates the background color of the cell. The default value is False.

Property	Meaning
`iscobol.export.excel.cell_ignore_borders (boolean)`	**True** = Ignores the border style of the cell. **False** = Replicates the border style of the cell. The default value is False.
`iscobol.export.excel.cell_locked (boolean)`	**True** = Mark cells as locked. **False** = Don't mark cells as locked. Locking cells has no effect until you protect the worksheet. This operation can be performed using external tools like Microsoft Excel. The default value is True.
`iscobol.export.excel.cell_numeric_format`	Specifies the number format for numeric cells. Refer to https://support.microsoft.com/en-us/kb/264372 for a list of valid values. There is no default.
`iscobol.export.excel.cell_wrap_text (boolean)`	**True** = Long text wraps in the cell. **False** = Long text is truncated. The default value is True.
`iscobol.export.excel.collapse_row_span (boolean)`	**True** = Collapse row span and avoid merging cells across rows. It implies iscobol.export.excel.remove_rows_space=true. **False** = Don't collapse row span and allow merging cells across rows. The default value is False.
`iscobol.export.excel.detect_cell_type (boolean)`	**True** = Preserve the type of the original Report field expressions and use it for the cell data type. **False** = Don't preserve the type of the original Report field expressions and don't use it for the cell data type. The default value is True.
`iscobol.export.excel.force_page_breaks (boolean)`	**True** = Create page breaks in the Excel sheet. **False** = Don't create page breaks in the Excel sheet. The default value is True.
`iscobol.export.excel.freeze_page_header (boolean)`	**True** = The Report page header is shown only on the top of the spreadsheet and is freezed during scrolling. When the content of the header changes, a new sheet is created. **False** = The Report page header is shown for each Report page exported in the spreadsheet. The default value is False.
`iscobol.export.excel.ignore_images (boolean)`	**True** = Ignore graphics and export only the text. **False** = Export both text and graphics. The default value is False.

Property	Meaning
`iscobol.export.excel.remove_columns_space (boolean)`	**0 =** Empty spaces that could appear between columns should not be removed. **1 =** Empty spaces that could appear between columns should be removed. **2 =** Empty spaces that could appear between columns should be removed. In the resulting output, useless columns are then removed as well. The default value is 0.
`iscobol.export.excel.remove_rows_space (boolean)`	**True =** Empty spaces that could appear between rows should be removed. **False =** Empty spaces that could appear between rows should not be removed. The default value is False.
`iscobol.export.excel.whitepage_background (boolean)`	**True =** Force cell white background. **False =** Don't force cell white background. The default value is True.

Report print

Property	Meaning
`iscobol.report.font.default`	This property has the same meaning and usage of iscobol.font.default *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.default * to represent the default font. If not set, then iscobol.font.default * is used also during Report print and preview.
`iscobol.report.font.fixed`	This property has the same meaning and usage of iscobol.font.fixed *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.fixed * to represent the fixed font. If not set, then iscobol.font.fixed * is used also during Report print and preview.
`iscobol.report.font.large`	This property has the same meaning and usage of iscobol.font.large *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.large * to represent the large font. If not set, then iscobol.font.large * is used also during Report print and preview.
`iscobol.report.font.medium`	This property has the same meaning and usage of iscobol.font.medium *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.medium * to represent the medium font. If not set, then iscobol.font.medium * is used also during Report print and preview.
`iscobol.report.font.small`	This property has the same meaning and usage of iscobol.font.small *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.small * to represent the small font. If not set, then iscobol.font.small * is used also during Report print and preview.
`iscobol.report.font.traditional`	This property has the same meaning and usage of iscobol.font.traditional *, It is considered by the IDE during Report print and preview, but not during Report painting. The Report Designer relies on iscobol.font.traditional * to represent the traditional font. If not set, then iscobol.font.traditional * is used also during Report print and preview.

Update Facility Configuration

Property	Meaning
`swupdater.items_list[.<os>[.<arch>]].<packageName>`	This property defines a list of items that will be included or excluded in the update process. You can specify file names or relative paths. Relative paths will be resolved according to the directory specified by swupdater.lib[.<os>[.<arch>]].<packageName>. Wildcards are accepted. Multiple items must be separated by comma. When specifying a path, the "/" file separator must be used, even on Windows. On platforms that are not Windows, the file names and paths specified by this property are case sensitive. Some examples: • A value of "file1,file" includes file1 and file2 in the list. • A value of "file1,mydir/*" includes file1 and all the files under mydir in the list. *os* is optional and can be any of the following values: win, linux, mac, solaris. If omitted, this property is considered for all clients regardless of their platform, otherwise the property is considered only for clients of a specific platform. *arch* is optional and can be either 32 or 64. If omitted, this property is considered for all clients regardless of their bitness, otherwise the property is considered only for clients of a specific bitness. Items list are supported only when swupdater.lib[.<os>[.<arch>]].<packageName> points to a directory. If it points to a ZIP file, items list can't be used. The inclusion and exclusion of items is controlled by the swupdater.exclude_items_list[.<os>[.<arch>]].<packagename> property.
`swupdater.exclude_items_list[.<os>[.<arch>]].<packagename>`	This property specifies how to treat the items listed by swupdater.lib[.<os>[.<arch>]].<packageName>. Possible values are: **-1** - ignore the items list and include all the files in the update process. **0** - include only the files specified by the items list in the update process and exclude all the others. **1** - include all the files except the ones specified by the items list in the update process. *os* is optional and can be any of the following values: win, linux, mac, solaris. If omitted, this property is considered for all clients regardless of their platform, otherwise the property is considered only for clients of a specific platform. *arch* is optional and can be either 32 or 64. If omitted, this property is considered for all clients regardless of their bitness, otherwise the property is considered only for clients of a specific bitness. The default value is -1.
`swupdater.version.<packageName>`	This property specifies the release of the package available on the server.

Property	Meaning
`swupdater.lib[.<os>[.<arch>]].<packageName>`	This property specifies the ZIP file or the directory containing the new version of the package. This entry can contain an absolute URL or a relative URL. *os* is optional and can be any of the following values: win, linux, mac, solaris. If omitted, this property is considered for all clients regardless of their platform, otherwise the property is considered only for clients of a specific platform. *arch* is optional and can be either 32 or 64. If omitted, this property is considered for all clients regardless of their bitness, otherwise the property is considered only for clients of a specific bitness. (swupdater.zipfile[.<os>[.<arch>]].<packageName> is supported for backward compatibility)

Client Configuration (isupdater.properties)

Property	Meaning
`swupdater.site`	This property specifies the HTTP URL to search for the updates. The HTTP server could be either an external HTTP Server like IIS or Apache or an isCOBOL Server started with -hs option. isUpdater will append "swupdater.properties" to this URL in order to download the "swupdater.properties" configuration file. For example, setting *swupdater.site=http://192.168.0.1:10996* will cause isUpdater to download the file "http://192.168.0.1:10996/swupdater.properties" via HTTP GET method.
`swupdater.logfile`	This property specifies the pathname of the log file where isUpdater traces its activity when logging is enabled. isUpdater environment variables can be used in the value of this property.
`swupdater.logging (boolean)`	**True =** Trace the isUpdater activity into the log file defined by *swupdater.logfile*. **False =** isUpdater activity is not traced. The default value is False.
`swupdater.jvm_options`	This property specifies the Java options to be used with the new JVM instantiated by isUpdater to run the program specified by the value of swupdater.mainclass. Multiple options must be separated by a space. For example, if you want the new JVM to use up to 500 MB of heap memory and allocate 256 MB for the metaspace, set: *swupdater.jvm_options=-Xmx500m -XX:MetaspaceSize=256m* Note - the creation of a new JVM is conditioned by the swupdater.new_jvm_always (boolean) configuration property.
`swupdater.mainclass`	This property specifies the class to run after the update checking. If this entry is missing then the isupdater tool terminates at the end of the update process.
`swupdater.net.ssl.trust_store`	If this property contains a path, then isUpdater will use an SSL connection, getting information about the sever certificate from the JKS keystore indicated by the path: a special value of '*' directs isUpdater to use the system Java keystore. isUpdater environment variables can be used in the value of this property.
`swupdater.net.ssl.trust_store_password`	This property specifies the password of the keystore when requested.
`swupdater.new_jvm_always (boolean)`	**True =** The program pointed by swupdater.mainclass is always executed in a new JVM. **False =** The program pointed by swupdater.mainclass is executed in a new JVM only if an update occurred. The default value is False.

Property	Meaning
`swupdater.http.ignore_certificates (boolean)`	**True =** If the handshaking fails due to invalid certificates, continue and connect anyway. **False =** If the handshaking fails due to invalid certificates, stop. The default value is False. Note - this property should be set to true only for test purposes. It's not good practice to ignore handshaking errors.
`swupdater.version.<packageName>`	This property specifies the release of the package <packageName> currently installed. This is free text, both letters and numbers are allowed. The version can be expressed in dotted form as usually software versions are expressed. If the expession between dots is a decimal number then it will be evaluated in such way, otherwise it will be evluated as alphabetical expression, for example: "10.4" is less than "10.10" and "10.4a" is greater than "10.10a". The isupdater tool will download new files only if the version specified by this property is less than the version available on the server.
`swupdater.directory.<packageName>`	This property specifies the directory in which the new software will be downloaded. If this entry is missing, then files are downloaded in the isupdater working directory. isUpdater environment variables can be used in the value of this property.
`swupdater.directory.clean.<packageName> (boolean)`	**True =** Clean the content of the client directory before downloading files from the server. Some jar libraries may not be removed as they're locked by the client JVM, in this case they are made zero bytes in size. **False =** Just download the files from the server to the client directory. **Note -** the client runtime must be version 2018 R1 or greater for this feature to work. Previous runtime versions don't support it. The default value is False.

isUpdater environment variables

The following variables can be used in the isUpdater client configuration:

Variable	Value
`${iscobol.home}`	The isCOBOL home directory. This path is calculated by removing the last directory from the path of the *isupdater.jar* library. For example, if *isupdater.jar* was loaded from '/opt/isCOBOL/lib', then *iscobol.home* will be '/opt/isCOBOL'.
`${java.home}`	The Java home directory.
`${user.home}`	The user's home directory.
`${user.temp}`	The user's temp directory.

For example, if you wish the log to be generated in the user's home directory, set:

```
swupdater.logfile=${user.home}/isupdater.log
```

isUpdater will internally translate "${user.home}/isupdater.log" to "C:\Users\UserName\isupdater.log" when running on Windows and "/home/UserName/isupdater.log" when running on Linux/Unix.

Library Routines Configuration

C$COPY

Property	Meaning
`iscobol.ccopy.client_temp_as_base_dir (boolean)`	**True** = In thin client, for client-side files, use the user's TEMP folder as base directory for relative file paths. **False** = In thin client, for client-side files, use the client's working directory as base directory for relative file paths. The default value is False.

C$EASYOPEN

Property	Meaning
`iscobol.easyopen.method`	This property specifies which method must be used by C$EASYOPEN to open the file. Possible values are: **JDIC** = use the JDIC component **JAVA** = use the java.awt.Desktop class **START** = use the START command on Windows **WINAPI** = call the ShellExecuteA Windows function. The default value is JAVA.

C$MYFILE

Property	Meaning
`iscobol.cmyfile.classname_only (boolean)`	**True** = only the class name (without path) is returned by C$MYFILE. **False** = a full file name (path and file name) is returned by C$MYFILE. The default value is False.

C$SOCKET

Property	Meaning
`iscobol.csocket.keepalive (boolean)`	**True** =The attribute SO_KEEPALIVE is turned on for sockets managed by the C$SOCKET routine. **False** = The attribute SO_KEEPALIVE is turned off for sockets managed by the C$SOCKET routine. The default value is False.
`iscobol.csocket.maxbuffersize`	Sets the SO_RCVBUF option value for sockets created by C$SOCKET. The default value is 0.

Property	Meaning
`iscobol.csocket.reuseaddr (boolean)`	**True** =The attribute SO_REUSEADDR is turned on for sockets managed by C$SOCKET routine. **False** = The attribute SO_REUSEADDR is turned off for sockets managed by C$SOCKET routine. The default value is False.
`iscobol.csocket.tcp_nodelay (boolean)`	**True** =socket packets are sent immediately. **False** = socket packets are delayed using the Nagle algorithm. The default value is True.

C$XML

Property	Meaning
`iscobol.cxml.indent_number`	This property changes the layout of XML files generated by C$XML. Possible values are: **-1** = all elements in the same line, no line feeds and no indentation. **0** = every element in a separate line, no indentation. **>0** = every element in a separate line, with a given indentation. The value of the property is the number of spaces used for the indentation. The default value is -1.

KEISEN

`iscobol.keisen.method`	This property specifies the line drawing method for the KEISEN routines. Valid values are 1 and 2. The default value is 1.

SYSTEM

Property	Meaning
`iscobol.system.exec`	This property specifies the name of a command to be executed when the SYSTEM routine is called. The parameter of the SYSTEM routine will be passed to the command. For example, setting *iscobol.system.exec=sh -c* and executing *CALL "SYSTEM" using "ls >out"* will cause the following command to be executed: *sh -c "ls >out"*. If this property is set to the special value "c" (lower case), the system() C routine will be used by isCOBOL in place of the Java API each time the SYSTEM library routine is called. In order to have the system() C routine available on Windows, the msvcrt library must be loaded in memory, therefore you should set also *iscobol.shared_library_list=msvcrt.dll* in the configuration. Programs compiled with the -cp option require the parameter passed to SYSTEM to be null terminated. Quotes may be used to isolate a command line parameter that includes spaces. When this property is set either to "c" or to the system command interpreter, quotes are stripped under Linux/Unix and preserved under Windows.

Utilities Configuration

COBFILEIO

Property	Meaning
`iscobol.cobfileio.efd_path`	Specifies the directory containing the EFD file

Property	Meaning
`iscobol.cobfileio.output_path`	Specifies the output directory where COBFILEIO will place the generated items
`iscobol.cobfileio.package`	Specifies the Java package name to be defined in the generated source code
`iscobol.cobfileio.use_resource_file`	If set to '1' or 'True', this setting causes all the strings to be generated with the '-r' prefix so that they will be loaded from resource files at run time. For a correct result, the following entries must be available in the resource file: number_too_large=*Number Too Large. Field* max_number=*Max Number* current_number=*Current Number* string_too_long=*String Too Long. Field* max_length=*Max Length* current_length=*Current Length* wrong_scale=*Wrong Scale. Field* max_scale=*Max Scale* current_scale=*Current Scale*

GIFE

`iscobol.gife.custom_font`	This property specifies the font used by GIFE's dialogs. It accepts a pair of font name and font size separated by dash. For example, in order to use Arial size 16, set: `iscobol.gife.custom_font=Arial-16` By default GIFE uses the label font set in the current Look & Feel.
`iscobol.gife.efd_directory`	This property specifies the default folder for EFD dictionaries. In the "Open File" dialog, GIFE automatically fills the "EFD File Path" field if: an EFD dictionary with the same name of the file exists in the folder where the opened file is located, or an EFD dictionary with the same name of the file exists in the folder set by *iscobol.gife.efd_directory*
`iscobol.gife.encrypt`	This property is considered for JIsam files. **True** = GIFE considers the file as encrypted. It uses the key specified by the property iscobol.file.encryption.key * to decrypt the file content. **False** = GIFE considers the file as plain. The default value is False.
`iscobol.gife.open_mode_io (boolean)`	**True** = GIFE opens the file for i/o as default **False** = GIFE opens the file for input as default The default value is False.
`iscobol.gife.rel_rec_size =n`	This property specifies the record size for the open of a relative file. *n* can be any positive number.
`iscobol.gife.num_convention=conv`	This property specifies the numeric convention used to represent numeric fields when a EFD dictionary is provided. conv can be any of the following: • -dca • -dcb • -dcd • -dcdm • -dci • -dcii • -dcm • -dcmi • -dcn • -dcr • -dcv The default value is -dca.

ISL

Property	Meaning
iscobol.isl.execute_debug (boolean)	**True** = The "-d" check-box will be checked in the ISL GUI. **False** =The "-d" check-box will not be checked in the ISL GUI. The default value is False.
iscobol.isl.java_options	This property specifies the options to be passed to the Java Runtime. If set, the corresponding field in the ISL GUI will be automatically filled with this value.
iscobol.isl.laf	This property specifies the LAF used by the launched program. If set, the corresponding field in the ISL GUI will be automatically filled with this value. Possible values are: • system • --system • metal • --metal • motif • --motif • GTK • --GTK • nimbus • --nimbus
iscobol.isl.nodisconnecterr (boolean)	**True** = The "-nodisconnecterr" check-box will be checked in the ISL GUI. **False** =The "-nodisconnecterr" check-box will not be checked in the ISL GUI. The default value is False.
iscobol.isl.prog_arguments	This property specifies the arguments for the COBOL program. If set, the corresponding field in the ISL GUI will be automatically filled with this value.
iscobol.isl.prog_name	This property specifies the name of the COBOL program. If set, the corresponding field in the ISL GUI will be automatically filled with this value.
iscobol.isl.updater_conf_file	This property specifies the configuration file for the isUpdater utility that is invoked when the user activates the -update option.

ISMIGRATE

Property	Meaning
iscobol.ctree.library	Set this property to “ctreestd” to use the c-tree standalone library (ctreestd) instead of the client/server library (ctree) for data migration. The standalone library is faster than the client/server library in reading and writing c-tree data. However, the standalone library is suitable only for data migration as it lacks of many features if compared with the client/server library (for example, file encryption is not supported). The use of this setting in a COBOL application not suggested. The default value is "ctree".
iscobol.ismigrate_additional_bytes	This property increases the output record length by the specified number of bytes.

Property	Meaning

iscobol.ismigrate_hook	This property specifies the name of a custom program that ISMIGRATE will call for each record read from the input file before writing the record to the output file. The feature allows you to alter the record content during the migration process.
	The hook program is called as a standard COBOL program so it must be available in the code-prefix or in the Classpath, depending on the current configuration.
	The program receives the following Linkage parameters:
	
	`01 INPUT-FULLNAME PIC X ANY LENGTH.`
	`01 INPUT-RECORD PIC X ANY LENGTH.`
	
	As the parameter names say, the former receives the full path name of the input file while the latter receives the content of the record read.
	The length of INPUT-RECORD shouldn't be changed by the hook program otherwise an unexpected result may occur since ISMIGRATE uses the original record length.
	
	If the hook program exits with a negative return code (e.g. GOBACK -1), then the record is skipped by ISMIGRATE.
	
	The hook program is not cancelled after being called.

iscobol.ismigrate_ignore_write_errors (boolean)	**True** = Continue the data migration even if write errors occur. **False** = Stop if a write error occurs. The default value is False.
iscobol.ismigrate_input_encrypt (boolean)	**True** = Consider the input files as encrypted. **False** = Don't consider the input files as encrypted. If the input files are JIsam, use iscobol.ismigrate_input_encryption_key to specify the encryption key. The default value is False.
iscobol.ismigrate_input_encryption_key	This property specifies the encryption key to be used when reading JIsam files during the migration process. It's evaluated when iscobol.ismigrate_input_encrypt (boolean) is true. It overrides iscobol.file.encryption.key * if both properties are set.
iscobol.ismigrate_input_file_index	This property defines the source file system. The possible values of iscobol.file.index are suitable also for this property.
iscobol.ismigrate_input_jdbc_driver	This property specifies the jdbc driver for the input files when migrating between two databases using "easydb" as ismigrate_input_file_index. If you're not migrating between two databases, but between a database and another file system, then set the standard iscobol.jdbc.driver .
iscobol.ismigrate_input_jdbc_url	This property specifies the jdbc url for the input files when migrating between two databases using "easydb" as ismigrate_input_file_index. If you're not migrating between two databases, but between a database and another file system, then set the standard iscobol.jdbc.url .
iscobol.ismigrate_logfile	This property specifies the path name of the ismigrate log file. By default, a file named "ismigrate.log" is created in the working directory.

Property	Meaning
iscobol.ismigrate_logging	**0** = The ISMIGRATE activity isn't traced and no dump files are generated. **1** = The ISMIGRATE activity is traced in the file pointed by ismigrate_logfile, but no dump files are generated. **2** = The ISMIGRATE activity is traced in the file pointed by ismigrate_logfile, and a dump file is generated for each migration that failed for one of these reasons: • a write error occurred and iscobol.ismigrate_ignore_write_errors (boolean) is set to true, • the hook program specified by iscobol.ismigrate_hook returned a negative exit code, • the verification performed due to iscobol.ismigrate_verify_records (boolean) set to true found one or more records that don't match. The log file contains information about the active configuration and the i/o operations. The dump files contain the hex content of the problematic records. The dump files are generated in the same directory as the log file and they have the same name of the input files that were not successfully migrated. The default value is 0.
iscobol.ismigrate_no_alphabet (boolean)	**True** = Don't pass the collating sequence to the output file handler. This is useful when migrating to file systems like Dci, which do not support collating sequences. **False** = Pass the collating sequence to the output file handler. The default value is False.
iscobol.ismigrate_no_directories (boolean)	**True** = Consider the first parameter (*InputFile*) as the name of the source file, and the second parameter (*OutputDir*) as the name of the destination file. Useful to migrate one file at a time. **False** = Consider the first parameter (*InputFile*) as a list of files, and the second parameter (*OutputDir*) as the name of the destination folder. Useful to migrate multiple files at a time. The default value is False.
iscobol.ismigrate_no_echo (boolean)	**True** = No output is printed on the system output. It might be useful in a scenario where you call ISMIGRATE from a COBOL program managing errors and record count via Linkage parameters without the need to have them printed on the console. **False** = Some information is printed on the system output. The default value is False.
iscobol.ismigrate_no_make (boolean)	**True** = Don't perform the build of the output file. This is useful when optimizing the migration of file systems like EasyDB, which create a table when opening a file. **False** = Build the output file. The default value is False.
iscobol.ismigrate_no_output_directory (boolean)	**True** = Ignore the second parameter. **False** = Consider the second parameter. This setting is useful when migrating to databases unless you wish to include a portion of the file path in the destination table name (e.g. if you set iscobol.easydb.dirlevel to value greater than zero). The default value is False.

Property	Meaning
`iscobol.ismigrate_open_extend (boolean)`	**True** = ISMIGRATE will open the output files in EXTEND mode. The output files must already exist, otherwise their opening will fail. **False** = ISMIGRATE will open the output files in OUTPUT mode, creating them if they don't exist or resetting them if they exist. The default value is False.
`iscobol.ismigrate_output_encrypt (boolean)`	**True** = The output files are created with the encrypted flag. Note that some file handlers ignore such flag. **False** = The encryption flag is not used to create the output files. If the output files are JIsam, use iscobol.ismigrate_output_encryption_key to specify the encryption key. The default value is False. *iscobol.ismigrate_make_encrypt* is still supported for backward compatibility.
`iscobol.ismigrate_output_encryption_key`	This property specifies the encryption key to be used when writing JIsam files during the migration process. It's evaluated when iscobol.ismigrate_output_encrypt (boolean) is true. It overrides iscobol.file.encryption.key * if both properties are set.
`iscobol.ismigrate_output_file_index`	This property defines the destination file system. The possible values of iscobol.file.index are suitable also for this property.
`iscobol.ismigrate_output_jdbc_driver`	This property specifies the jdbc driver for the output files when migrating between two databases using "easydb" as ismigrate_output_file_index. If you're not migrating between two databases, but between a database and another file system, then set the standard iscobol.jdbc.driver .
`iscobol.ismigrate_output_jdbc_url`	This property specifies the jdbc url for the output files when migrating between two databases using "easydb" as ismigrate_output_file_index. If you're not migrating between two databases, but between a database and another file system, then set the standard iscobol.jdbc.url .
`iscobol.ismigrate_remove_extension`	This property removes the extension from the incoming data file name before processing it. Because some data file systems automatically add a file extension to the filename (JIsam and c-tree for instance), you can use this property to remove an existing extension to avoid a file not found error. For example, ISMIGRATE will ignore the .dat extension of a passed file name "filename.dat" if *iscobol.ismigrate_remove_extension=dat* is set.
`iscobol.ismigrate_strip_extension (boolean)`	**True** = Strip the extension from the file name passed to ISMIGRATE before naming the output file. Set this to avoid a double extension created when the file extension automatically added by the output file handler would be added to the input file extension. **False** = Use the full file name passed to ISMIGRATE when naming the output file. The default is False.

Property	Meaning
`iscobol.ismigrate_verify_records (boolean)`	**True** = Check if the records in the output file match with the records in the input file after the file has been migrated. **False** = Don't check if the records in the output file match with the records in the input file after the file has been migrated. The default value is False.

JDBC2FD

Property	Meaning
`iscobol.jdbc2fd.current_schema_only`	**True** = Show only tables of the current schema. **False** = Show all tables returned by the database metadata. The default value is False.

isCOBOL Code Coverage Configuration

The following property must be set when the runtime starts and are evaluated if the -coverage option is used.

Property	Meaning
`iscobol.coverage.analysis.excludes`	This property specifies a comma separated list of COBOL classes that must be excluded from the analysis. Regular expressions can be used to specify a pattern, for example: `iscobol.coverage.analysis.excludes=SAMPLE,GUI.*` will exclude the SAMPLE class and all the COBOL classes whose name starts with "GUI". By default, all COBOL classes are analyzed.
`iscobol.coverage.analysis.includes`	This property specifies a comma separated list of COBOL classes that must be included in the analysis. Regular expressions can be used to specify a pattern, for example: `iscobol.coverage.analysis.includes=SAMPLE,IO.*` will include the SAMPLE class and all the COBOL classes whose name starts with "IO". By default, all COBOL classes are analyzed.

Property	Meaning
`iscobol.coverage.append`	This property specifies if the coverage report must be merged with existing reports or not. It affects only XML reports, so it's considered only if iscobol.coverage.xml is set. Possible values are: **0** = if the XML file already exists, overwrite it **1** = if the XML file already exists, merge the contents of the existing XML with the content of the current XML. This property can also be set to a comma-separated list of existing XML reports, e.g. `iscobol.coverage.append=C:\\Reports\\report1.xml,C:\\Reports\\report2.xml` In such case, the resulting XML will be obtained by merging all the XML files in the list with the new coverage report. The default value is 0.
`iscobol.coverage.classfiles`	This property specifies the list of paths where the Coverage engine can find the class files used to build the report. Multiple paths must be separated by the line feed character or by the current operating system path separator. By default the report is built using the classes loaded by the runtime.
`iscobol.coverage.html`	This property specifies the directory where the Coverage engine generates the HTML report. If the directory doesn't exist, it's automatically created. The default value is "./htmlReport". The default folder "./htmlReport" is created only if iscobol.coverage.xml is not set.
`iscobol.coverage.sessionname`	This property specifies the name of the session. This name appears on top of the index.html file of the HTML report. The default value is the name of the main program as specified on the command-line.
`iscobol.coverage.sourcefiles`	This property specifies the list of paths where the Coverage engine can find the COBOL sources. Multiple paths must be separated by the line feed character or by the current operating system path separator. The default value is "." (the current directory).
`iscobol.coverage.xml`	This property specifies the pathname of a file that will host the coverage report in XML format. If this property is not set, then the XML report is not generated. If this property is set but iscobol.coverage.html is not set, then only the XML report is generated.

Unit Test Configuration

The following property must be set when the runtime starts and are evaluated if the -iut option is used.

Property	Meaning
`iscobol.unit_test.html`	This property specifies the directory where the Unit Test engine generates the HTML report. If the directory doesn't exist, it's automatically created. The default value is "./htmlReport". The default folder "./htmlReport" is created only if iscobol.unit_test.xml is not set. If the Unit Test is performed along with Code Coverage, then this property is ignored and the report is generated in the folder specified by iscobol.coverage.html.
`iscobol.unit_test.list_file`	This property specifies the pathname of one or more text files that includes the list of programs to be included in the test. Multiple pathnames must be separated by \n character sequence or by the current operating system path separator. Within these files, each program must be indicated on a separate line. The runtime takes care of normalizing the program name, stripping the extension, making it upper-case and replacing dashes with underscores, same as it happens on the runtime command line. If this property points to an invalid file, like a file that doesn't exist, then an empty report is generated.
`iscobol.unit_test.xml`	This property specifies the pathname of a file that will host the coverage report in XML format. If this property is not set, then the XML report is not generated. If this property is set but iscobol.unit_test.html is not set, then only the XML report is generated.

Profiler Configuration

The following properties must be set when the runtime starts and are evaluated if the -profile option is used.

Property	Meaning
`iscobol.profiler.excludes`	This property specifies the list of COBOL classes that must not be analyzed by the profiler. Multiple values must be separated by comma. Regular expressions can be used to specify a pattern, for example: `iscobol.profiler.excludes=SAMPLE,GUI.*` will exclude the SAMPLE class and all the COBOL classes whose name starts with "GUI". By default, all COBOL classes are profiled.
`iscobol.profiler.html`	This property specifies the directory where the profiler generates the HTML report. If the directory doesn't exist, it's automatically created. The default value is "./hprofHtmlReport". The default folder "./hprofHtmlReport" is created only if iscobol.profiler.xml is not set.

Property	Meaning
`iscobol.profiler.includes`	This property specifies the list of COBOL classes that must be analyzed by the profiler. Multiple values must be separated by comma. Regular expressions can be used to specify a pattern, for example: `iscobol.profiler.includes=SAMPLE,IO.*` will include the SAMPLE class and all the COBOL classes whose name starts with "IO". By default, all COBOL classes are profiled.
`iscobol.profiler.xml`	This property specifies the pathname of a file that will host the profiler report in XML format. If this property is not set, then the XML report is not generated. If this property is set but iscobol.profiler.html is not set, then only the XML report is generated.

Internal Objects Configuration

JSONSTREAM

Property	Meaning
`iscobol.jsonstream.allow_backslash_escaping_any_character (boolean)`	**True** = When the JSONStream Class (com.iscobol.rts.JSONStream) reads a JSON stream, backslash is allowed before any character. **False** = When the JSONStream Class (com.iscobol.rts.JSONStream) reads a JSON stream, backslash is allowed only in escapes: \b, \f, \n, \r, \t, \" and \\. The default value is False.
`iscobol.jsonstream.indent_number`	This property specifies the number of columns for the indentation of items in JSON files generated by the JSONStream Class (com.iscobol.rts.JSONStream). When this property is set to a value of 0 or grater, each element is generated on a separate line, that means your stream will include CRLF. The default value is -1, that disables indentation and allows you to obtain a single line stream without CRLF.
`iscobol.jsonstream.initialize_on_read (boolean)`	**True** = the group item that will host the data read from JSON is automatically initialized before reading data from JSON. **False** = the group item that will host the data read from JSON is not automatically initialized before reading data from JSON. You can initialize it with an INITIALIZE statement. The default value is False.
`iscobol.jsonstream.omit_empty_elements (boolean)`	**True** = empty elements are not generated by the JSONStream Class (com.iscobol.rts.JSONStream) **False** = empty elements are generated by the JSONStream Class (com.iscobol.rts.JSONStream) An element is considered empty when it's alphanumeric and it's 0 bytes in size. This condition can occur if the underlying data item is an initialized ANY LENGTH item or if the value has been trimmed by iscobol.jsonstream.rtrim (boolean). The default value is True.
`iscobol.jsonstream.rtrim (boolean)`	**True** = remove trailing spaces from JSON items value **False** = keep trailing spaces in JSON items value The default value is False. This property is evaluated before iscobol.jsonstream.omit_empty_elements (boolean).

Property	Meaning
`iscobol.xmlstream.indent_number`	This property specifies the number of columns for the indentation of items in XML files generated by the XMLStream Class (com.iscobol.rts.XMLStream). When this property is set to a value of 0 or grater, each XML element is generated on a separate line, that means your XML stream will include CRLF. The default value is -1, that disables indentation and allows you to obtain a single line XML stream without CRLF.
`iscobol.xmlstream.initialize_on_read (boolean)`	**True** = the group item that will host the data read from XML is automatically initialized before reading data from XML. **False** = the group item that will host the data read from XML is not automatically initialized before reading data from XML. You can initialize it with an INITIALIZE statement. The default value is False.
`iscobol.xmlstream.omit_empty_elements (boolean)`	**True** = empty elements are not generated by the XMLStream Class (com.iscobol.rts.XMLStream) **False** = empty elements are generated by the XMLStream Class (com.iscobol.rts.XMLStream) An element is considered empty when it's alphanumeric and it's 0 bytes in size. This condition can occur if the underlying data item is an initialized ANY LENGTH item or if the value has been trimmed by iscobol.xmlstream.rtrim (boolean). The default value is True.
`iscobol.xmlstream.resolve_references (boolean)`	**True** = resolve href/id references in the read XML stream **False** = don't resolve href/id references in the read XML stream The default value is False.
`iscobol.xmlstream.rtrim (boolean)`	**True** = remove trailing spaces from XML values **False** = keep trailing spaces in XML values The default value is False. This property is evaluated before iscobol.xmlstream.omit_empty_elements (boolean).

Keyboard Configuration

(*) The asterisk after the property name means that the property is static, it's inquired only once by the

Framework and then changing it using the SET ENVIRONMENT statement has no effect.

Property	Meaning
`iscobol.kbd_auto_return`	When set to a non-zero value, this property specifies the termination value that is stored into crt status when ACCEPT terminates automatically due to a AUTO-TERMINATE clause. The default value is 0. (iscobol.keyboard.kbd_auto_return is still supported for backward compatibility)

`iscobol.key.accepted_control_characters`	Specifies which control characters should be accepted and displayed by Entry-Field, Combo-Box and Grid as well as character-based input fields.
	A control character or non-printing character (NPC) is a code point in a character set, that does not represent a written symbol. Control characters are used as signaling to cause effects other than the addition of a symbol to the text.
	
	Set this property to the list of control characters using hex notation.
	For example, in order to allow 0x1D and 0x1E to be inserted in an Entry-Field, Combo-Box or Grid, set the property as follows in the configuration file:
	
	`iscobol.key.accepted_control_characters=\u001d\u001e`
	
	The property can also be set dynamically by COBOL programs with this syntax:
	
	`set environment "key.accepted_control_characters" to x"1d1e"`
	
	Note - The control characters are usually discarded because it is assumed that they are already used for different actions than displaying a character in an input field. For example, the character 0x08 (backspace) deletes a character in the text component. Adding 0x08 to the list of accepted control characters would cause the backspace to display an odd symbol in the field instead of deleting a character. Use this property carefully in order to avoid side effects.

`iscobol.key.default_shortcuts_enabled (boolean) *`	**True =** Copy/Cut/Paste/Undo/Redo (Ctrl+C/X/V/Z/Y) actions are enabled on all controls, so their exception codes can't be caught by the program.
	False = Copy/Cut/Paste/Undo/Redo (Ctrl+C/X/V/Z/Y) actions are disabled and their exception codes can be caught by the program.
	
	When the property is set to False, the Copy/Cut/Paste/Undo/Redo (Ctrl+C/X/V/Z/Y) actions can be enabled using this code:
	
	`SET ENVIRONMENT "KEYSTROKE" TO "Exception=101 ^C".`
	`SET ENVIRONMENT "KEYSTROKE" TO "Exception=102 ^X".`
	`SET ENVIRONMENT "KEYSTROKE" TO "Exception=103 ^V".`
	`SET ENVIRONMENT "KEYSTROKE" TO "Exception=104 ^Z".`
	`SET EXCEPTION 101 TO copy-selection`
	`SET EXCEPTION 102 TO cut-selection`
	`SET EXCEPTION 103 TO paste-selection`
	`SET EXCEPTION 104 TO UNDO`
	
	However, they affect only the Entry-Field and Combo-Box controls in this case.
	
	The default value is True.

Property	Meaning
`iscobol.key.KeyName`	This property specifies the keyboard configuration. See below for further details.
`iscobol.key.*f1.system (boolean)`	**True** = The keystroke CTRL+F1 is not intercepted by the isCOBOL Framework and affects the COBOL window directly. **False** = The keystroke CTRL+F1 is intercepted by the isCOBOL Framework and doesn't affect the COBOL window, but it can be handled by the COBOL program. The default value is True.
`iscobol.key.@f4.system (boolean)`	**True** = The keystroke ALT+F4 is not intercepted by the isCOBOL Framework and affects the COBOL window directly. **False** = The keystroke ALT+F4 is intercepted by the isCOBOL Framework and doesn't affect the COBOL window, but it can be handled by the COBOL program. The default value is True.
`iscobol.key.f4.system (boolean)`	**True** = If the focus is either on a combo-box or on a paged list-box, then the F4 key produces a system action (e.g. drop the combo-box or trigger the list-box search respectively) and doesn't generate an exception in the current ACCEPT. **False** = The key F4 always generates an exception in the current ACCEPT. The default value is True. (iscobol.gui.f4_drops_combobox is still supported for backward compatibility)
`iscobol.key.f10.system (boolean)`	**True** = If the window where the current ACCEPT is performed includes a menu bar, pressing F10 activates the menu bar. **False** = The key F10 always generates an exception in the current ACCEPT. The default value is True.
`iscobol.keystroke.firstlast_on_screen (boolean)*`	**True** = *edit=first* and *edit=last* allow to move the cursor to the first or last field of the screen respectively. **False** = *edit=first* and *edit=last* have no effect. The default value is False.
`iscobol.keystroke.updown_like_prevnext (boolean) *`	**True** = *edit=up* has the same effect of *edit=previous* and *edit=down* has the same effect of *edit=next* in key settings. **False** = key settings preserve their behavior. The default value is False. This property affects the ACCEPT of user input on both character based and GUI screens. When set to true, the No-Group-Tab style is assumed for every RADIO-BUTTON.

Function and special keys can be configured using the corresponding property.

Property	Key
`iscobol.key.enter`	Enter
`iscobol.key.tab`	Tab

Property	Key
iscobol.key.escape	Esc
iscobol.key.backspace	Backspace
iscobol.key.end	End
iscobol.key.home	Home
iscobol.key.insert	Ins
iscobol.key.delete	Del
iscobol.key.clear	Clear
iscobol.key.help	Help
iscobol.key.left	Left
iscobol.key.right	Right
iscobol.key.up	Up
iscobol.key.down	Down
iscobol.key.pageup	PageUp
iscobol.key.pagedown	PageDown
iscobol.key.f1	F1
iscobol.key.f2	F2
iscobol.key.f3	F3
iscobol.key.f4	F4
iscobol.key.f5	F5
iscobol.key.f6	F6
iscobol.key.f7	F7
iscobol.key.f8	F8
iscobol.key.f9	F9
iscobol.key.f10	F10
iscobol.key.f11	F11
iscobol.key.f12	F12
iscobol.key.f13	F13
iscobol.key.f14	F14
iscobol.key.f15	F15

Property	Key
iscobol.key.f16	F16
iscobol.key.f17	F17
iscobol.key.f18	F18
iscobol.key.f19	F19
iscobol.key.f20	F20
iscobol.key.divide	/ (NumPad)
iscobol.key.multiply	* (NumPad)
iscobol.key.subtract	- (NumPad)
iscobol.key.add	+ (NumPad)
iscobol.key.decimal	. (NumPad)
iscobol.key.numpad0	0 (NumPad)
iscobol.key.numpad1	1 (NumPad)
iscobol.key.numpad2	2 (NumPad)
iscobol.key.numpad3	3 (NumPad)
iscobol.key.numpad4	4 (NumPad)
iscobol.key.numpad5	5 (NumPad)
iscobol.key.numpad6	6 (NumPad)
iscobol.key.numpad7	7 (NumPad)
iscobol.key.numpad8	8 (NumPad)
iscobol.key.numpad9	9 (NumPad)
iscobol.key.mmov	mouse moved
iscobol.key.mldw	left mouse button pressed
iscobol.key.mlup	left mouse button released
iscobol.key.mldc	left mouse button double-clicked
iscobol.key.mmdw	middle mouse button pressed
iscobol.key.mmup	middle mouse button released
iscobol.key.mmdc	middle mouse button double-clicked
iscobol.key.mrdw	right mouse button pressed
iscobol.key.mrup	right mouse button released

Property	Key
`iscobol.key.mrdc`	right mouse button double-clicked

The characters "^", "*" and "@" represent the [Shift], [Ctrl] and [Alt] key, respectively. Put one or more of them after the key name to define key combinations.

For example, [F1] is iscobol.key.f1, [Shift+F1] is iscobol.key.^f1, [Ctrl+F1] is iscobol.key.*f1 and [Alt+F1] is iscobol.key.@f1. Any combination of "^", "*" and "@" is valid, [Shift+Ctrl+Alt+F1] is iscobol.key.^*@f1.

Letter and number keys can be configured using properties in the form *iscobol.key.*# (where # is a letter from 'a' to 'z' or a number from '0' to '9'), and they can be configured in conjunction with Alt, Ctrl or Shift. For example:

- setting iscobol.key.a allows you to configure the keystroke [A]
- setting iscobol.key.*a allows you to configure the keystroke [Ctrl+A]
- setting iscobol.key.@a allows you to configure the keystroke [Alt+A]
- setting iscobol.key.^a allows you to configure the keystroke [Sift+A]
- setting iscobol.key.^*@a allows you to configure the keystroke [Shift+Ctrl+Alt+A]

Note - iscobol.key.@ followed by a number from '0' to '9' assigns an exception value that terminates the accept only when the number was entered from the standard keyboard, not from the numpad. This separates these key combinations from the Alt+<Numpad number> combinations used to enter special characters.

Pressing either the "Ctrl" key or the "Shift+Ctrl" key combination in conjunction with a key letter (A-Z) causes an exception whose number varies from 1 (if the letter is A) to 26 (if the letter is Z) by default. The behavior of these key combinations can be redefined using the properties iscobol.key.*a through iscobol.key.*z and iscobol.key.^*a through iscobol.key.^*z. However, this kind of keystroke might be intercepted by the current graphical control and don't cause any exception. For example, when the focus is on an ENTRY-FIELD, pressing Ctrl+B causes an exception whose number is 2, while pressing Ctrl+C doesn't cause an exception whose number is 3, it copies the selected text to the clipboard instead. To make all Ctrl+letter combinations cause an exception and lose copy, cut, paste, undo, redo functions on GUI controls, set iscobol.key.default_shortcuts_enabled (boolean) * to false in the configuration.

iscobol.key.*KeyName* can be set to one or more of the following values:

`data=`*`Value`*	*Value* represents the character sent to the program.
`exception=`*`Value`*	*Value* is a numeric value from 1 to 65536 representing the exception value generated for the program.
`termination=`*`Value`*	*Value* is a numeric value from 1 to 65536 representing the termination value generated for the program.
`hotkey=`*`Value`*	*Value* is the program to be called.

`edit=Value`	*Value* represents how the cursor is moved within controls belonging to the same Screen Section. Possible values are: **next =** the cursor goes to the next control **previous =** the cursor goes to the previous control **first =** the cursor goes to the first control [A] **last =** the cursor goes to the last control [A] **up =** the cursor goes to the nearest control above the current one **down =** the cursor goes to the nearest control below the current one **pageup =** the cursor moves as if Page-Up was pressed **pagedown =** the cursor moves as if Page-Down was pressed **backspace =** the cursor moves back by one digit deleting the character **delete =** the digit next to the cursor is deleted **insert =** the insert mode is changed **clear =** the field content is erased and the cursor is placed at the beginning of the field. **cl2end =** the content from the current cursor position to the end of the field is erased **erase-all =** all fields controlled by the ACCEPT statement are erased and the cursor is moved to the home position of the first field. This value should not be used in conjunction with *exception* or *termination*.

`search=Context`	*Context* represents where the keystroke will trigger a search. Possible values are:
	
	grid = the Grid control
	print-preview = the Print Preview dialog
	tree-view = the Tree-View control
	web-browser = the Web-Browser control
	
	You can specify one or more contexts, separated by comma.
	
	By default, the following setting is active: *iscobol.key.*f=search=grid,list-box,print-preview,tree-view,web-browser*, so the search is always triggered by Ctrl-F.
	Let's assume for example that you don't want the search with Ctrl-F on Grid and you want the search with Alt-F on the print preview; then you would set
	`iscobol.key.*f=search=list-box,tree-view,web-browser`
	`iscobol.key.@f=search=print-preview`

[A] Supported only if iscobol.keystroke.firstlast_on_screen (boolean)* is set to true.

Note - Function and special keys cannot be configured to send characters to the program, the data= setting has no effect for them.

Note - On graphical screens, if the active graphical control intercepts some keys for its own functionality, these keys are not returned to the COBOL program and therefore they don't raise any exception. This is the reason why for example F5 and ESC are not intercepted by the COBOL program while the focus is on a web-browser.

Default Keyboard Configuration

Property	Value
iscobol.key.enter	termination=13
iscobol.key.tab	termination=9 edit=next
iscobol.key.^tab	edit=previous
iscobol.key.escape	exception=27
iscobol.key.end	edit=last
iscobol.key.home	edit=first
iscobol.key.help	exception=90
iscobol.key.left	edit=left
iscobol.key.right	edit=right
iscobol.key.up	exception=52 edit=previous[A]
iscobol.key.down	exception=53 edit=next[A]
iscobol.key.pageup	exception=67 edit=pageup
iscobol.key.pagedown	exception=68 edit=pagedown
iscobol.key.backspace	edit=backspace
iscobol.key.insert	edit=insert
iscobol.key.delete	edit=delete
iscobol.key.f1	exception=1
iscobol.key.f2	exception=2
iscobol.key.f3	exception=3
iscobol.key.f4	exception=4 or none, depending on iscobol.key.f4.system (boolean)
iscobol.key.f5	exception=5
iscobol.key.f6	exception=6
iscobol.key.f7	exception=7
iscobol.key.f8	exception=8
iscobol.key.f9	exception=9
iscobol.key.f10	exception=10 or none, depending on iscobol.key.f10.system (boolean)
iscobol.key.f11	exception=11
iscobol.key.f12	exception=12

Property	Value
iscobol.key.f13	exception=13
iscobol.key.f14	exception=14
iscobol.key.f15	exception=15
iscobol.key.f16	exception=16
iscobol.key.f17	exception=17
iscobol.key.f18	exception=18
iscobol.key.f19	exception=19
iscobol.key.f20	exception=20
iscobol.key.divide	data=/
iscobol.key.multiply	data=*
iscobol.key.subtract	data=-
iscobol.key.add	data=+
iscobol.key.decimal	data=, or data=. depending on the current locale
iscobol.key.numpad0	data=0
iscobol.key.numpad1	data=1
iscobol.key.numpad2	data=2
iscobol.key.numpad3	data=3
iscobol.key.numpad4	data=4
iscobol.key.numpad5	data=5
iscobol.key.numpad6	data=6
iscobol.key.numpad7	data=7
iscobol.key.numpad8	data=8
iscobol.key.numpad9	data=9
iscobol.key.mmov	exception=80
iscobol.key.mldw	exception=81
iscobol.key.mlup	exception=82
iscobol.key.mldc	exception=83
iscobol.key.mmdw	exception=84
iscobol.key.mmup	exception=85

Property	Value
iscobol.key.mmdc	exception=86
iscobol.key.mrdw	exception=87
iscobol.key.mrup	exception=88
iscobol.key.mrdc	exception=89

[A]With the default configuration *key.up* and *key.down* behaves as follows: if there are fields below the current cursor location, the cursor moves to the one on the closest lower line. If there is more than one field on that line, the cursor moves to the one closest to its current horizontal location. The cursor will try to stay in the same column. If there are no fields below the current line, then no action is taken unless an EXCEPTION or TERMINATION value has been assigned, in this case *key.up* and *key.down* act as termination keys.

Acucobol-GT key codes

The Acucobol-GT KEYSTROKE configuration property is supported for compatibility. You can set this property in the program using a Format 6 SET statement or using the C$KEYSTROKE routine. Setting the property in a configuration file has no effect.

The following key codes are supported:

Key Code	Description
ZB	Backspace
8	Backspace
9	Tab
13	Enter
27	Escape
127	Delete
^M	Enter
^H	Backspace
^I	Tab
^[	Escape
^A - ^Z	Ctrl+A - Ctrl+Z
A0 - A9	Ctrl+0 - Ctrl+9
k1 - k10	F1 - F10
kd	Down
kh	Home

Key Code	Description
kl	Left
kr	Right
ku	Up
kA	Insert
kB	Shift+Tab
kE	Ctrl+End
kL	Ctrl+Delete
kN	Pag Down
kP	Pag Up
K1 - K0	Shift+F1 - Shift+F10
KB	Ctrl+Pag Down
KC	Ctrl + Home
KE	End
KI	Insert
KT	Ctrl+Pag Up
KX	Delete
Kd	Ctrl+Down
Kl	Ctrl+Left
Kr	Ctrl+Right
Ku	Ctrl+Up
K?	Help
S1 - S0	Shift+Ctrl+F1 - Shift+Ctrl+F10
U1 - U2	F11 - F12
U3 - U4	Shift+F11 - Shift+F12
U5 - U6	Ctrl+F11 - Ctrl+F12
U7 - U8	Alt+F11 - Alt+F12
U9 - U0	Shift+Ctrl+F11 - Shift+Ctrl+F12
Mv	mouse moved
Ml	left mouse button down

Key Code	Description
ML	left mouse button up
M1	left mouse button double clicked
Mm	middle mouse button down
MM	middle mouse button up
M2	middle mouse button double clicked
Mr	right mouse button down
MR	right mouse button up
M3	right mouse button double clicked

For example, in isCOBOL you can assign an exception value of 100 the the Backspace key in two ways:

- Standard isCOBOL way

```
set environment "key.backspace" to "exception=100"
```

- Acucobol compatible way

```
set environment "keystroke" to "exception=100 ZB"
```

Acucobol-GT key codes for W$KEYBUF routine

The following key codes can be used with the W$KEYBUF routine:

Key	Key Code
Alt+A	KA or @A
Alt+B	AB or @B
Alt+C	AC or @C
Alt+D	KD or @D
Alt+E	Kx or @E
Alt+Equals	A=
Alt+F	KF or @F
Alt+F1 - F10	a1 - a0
Alt+F11	U7
Alt+F12	U8
Alt+G	@G or AG

Key	Key Code
Alt+H	@H or K?
Alt+I	@I
Alt+J	@J or AJ
Alt+K	@K
Alt+L	@L or KL
Alt+M	@M or KM
Alt+Minus	A-
Alt+N	@N or AN
Alt+Numpad 1 - Numpad 0	A1 - A0
Alt+O	@O or AO
Alt+P	@P or KP
Alt+Q	@Q or AQ
Alt+R	@R or KR
Alt+S	@S or KS
Alt+T	@T or AT
Alt+U	@U or AU
Alt+V	@V or KV
Alt+W	@W or AW
Alt+X	@X
Alt+Y	@Y or AY
Alt+Z	@Z or AZ
Backspace	ZB
Cancel	Kc
Ctrl+A - H	^A - ^H
Ctrl+Backslash	^\
Ctrl+Backspace	127
Ctrl+Circumflex	^^
Ctrl+Closed bracket	^]
Ctrl+Delete	kL

Key	Key Code
Ctrl+Down arrow	Kd
Ctrl+End	kE
Ctrl+F1 - F10	C1 - C0
Ctrl+F11	U5
Ctrl+F12	U6
Ctrl+Home	KC
Ctrl+Insert	kA
Ctrl+J - L	^J - ^L
Ctrl+Left arrow	Kl
Ctrl+Left arrow	Kr
Ctrl+N - Z	^N - ^Z
Ctrl+Page down	KB
Ctrl+Pageup	KT
Ctrl+Shift+F1 - F10	S1- S0
Ctrl+Shift+F11	U9
Ctrl+Shift+F12	U0
Ctrl+Underscore	^_
Ctrl+Up arrow	Ku
Delete	KX
Down arrow	kd
End	KE
Enter	^M
Escape	^[
F1 - F10	k1 - k0
F11	U1
F11 - F20	K1 - K0
F12	U2
Home	kh
Insert	Kl

Key	Key Code
Left arrow	kl
Page down	kN
Page up	kP
Right arrow	kr
Shift+F11	U3
Shift+F12	U4
Shift+Tab	kB
Tab	^I
Up arrow	ku

RM/COBOL key codes for C$MBAR, C$RBMENU and C$TBAR routines

The following key codes can be used with the W$KEYBUF routine:

Key Code	Key
\a	Alt
\b	Backspace
\\	Bacslash character
\c	Control
\d	Delete
\e	Escape
\f0	F10
\f1	F1
\f2	F2
\f3	F3
\f4	F4
\f5	F5
\f6	F6
\f7	F7
\f8	F8
\f9	F9

Key Code	Key
\fa	F10
\fb	F11
\fc	F12
\g	AltGr
\i	Insert
\n	New line
\p	Pause
\qa	ATTN
\qc	Caps Lock
\qp	PA1
\s	Shift
\t	Tab
\wa	Applications
\wc	CRSEL
\we	EREOF
\wl	Left Windows Logo
\wp	PLAY
\wr	Right Windows Logo
\wx	EXSEL
\x	Exit program
\zb	Begin
\zc	Clear
\zd	Down Arrow
\ze	End
\zh	Home
\zl	Left Arrow
\zm	ZOOM
\zn	Page Down
\zp	Page Up

Key Code	Key
\zr	Right Arrow
\zs	Scroll Lock
\zu	Up Arrow
\z9	Num Lock

Chapter 2

Debugger

Overview

isCOBOL provides a visual source-level debugger. In order to debug programs, they must be compiled with either -d option or -dx option. For example:

```
iscc -d Options SourceCode
```

The following command starts a debugging session of the *ProgramName* program.

```
iscrun -d ProgramName
```

When running on Windows, the following command can also be used:

```
isrun -d ProgramName
```

Note - The Debugger takes advantage of some Compiler features, therefore the isCOBOL Compiler must be installed and licensed on the machine where the above commands are used.

Debugging a multi-thread program

When the debugged program generates more threads, it's possible to switch from one thread to another by choosing the desired thread at the bottom of the *Run* menu.

While the debugger is waiting for user input, all threads are blocked. When the debugger gives the control to the program, all threads run.

Remote Debugging

isCOBOL's Debugger can be used in remote debugging scenarios, a benefit in these situations when you want to debug:

- to debug a servlet or a web service running under Tomcat or other servlet container,
- to debug a COBOL program running via webclient,
- to debug a COBOL program that is called by foreign languages like C or Java,
- to debug a program with character-based interface managed by CHARVA,
- to debug a remote program running in the isCOBOL Server.

Here are the basic steps to activating the remote debugger to debug a program in one of the above situations.

1. Compile the program(s) to be debugged with either -d option or -dx option.

2. Set the iscobol.rundebug * property in your run environment to either value 1 or 2:

iscobol.rundebug=1	The first program will start and run immediately. The remote debug session waits and listens for a program that is compiled for debug to launch, connecting to the program when it detects it. This is useful when you only want to debug some of your application's programs.
iscobol.rundebug=2	The runtime framework will start in debug mode, pausing to connect to a remote debugger before running the program. This is useful when all your programs are compiled for debug and you want to start your remote debugger at the very first line of the application.

3. Start the remote debugger session on your desktop.

To start a remote debugging session, the isCOBOL Debugger needs to know the host name of the machine running the program to be debugged and the port number dedicated to the debugger connection. Use the following command:

```
iscrun [ -d ] -r [ [ HostName ]  Port ]
```

When running on Windows, the following command can also be used:

```
isrun [ -d ] -r [ [ HostName ]  Port ]
```

- The -d option can be omitted when -r is used.
- If *HostName* is omitted, then localhost is assumed.
- If *Port* is omitted, then 9999 is assumed.

Note - The debugger listens and runs on the TCP/IP port 9999. A different port number can be set via the the iscobol.debug.port configuration property.

Note - In thin client environment you don't need to use the remote debugger. You can debug the isCOBOL Server activity by running the isCOBOL Client with the -d option as explained in Usage of isCOBOL Client, Format 6.

Debugging Programs compiled with isCOBOL 2020R1 or previous versions

Prior to isCOBOL 2020 R2, the Compiler didn't store the source code into the compiled class file, so it was necessary to direct the Debugger to the location of the source files. The Debugger loaded the source files separately while attaching a class.

If your programs have been compiled by an isCOBOL version 2020 R1 or previous, then it's still necessary to direct the Debugger to the location of the source files. This is done through the iscobol.debug.code_perfix configuration property, which can be passed on the command line as follows:

```
iscrun [ -J-Discobol.debug.code_prefix=SourcePaths ][ -d ] -r [ [ HostName ]  Port ]
```

Where *SourcePaths* is a list of paths separated by the system path separator (";" on Windows, ":" on Unix).

If iscobol.debug.code_prefix is not set, then the Debugger looks for source files in the Classpath.

Remote debugging programs compiled with isCOBOL 2020 R1 or previous versions

While debugging a remote runtime, the Debugger looks for source files on the local PC.

If the source files are on the remote machine instead, you can ask the remote runtime to send the source files through TCP/IP to the client. In order to activate this feature, set iscobol.debug.remote_source to true on the local PC and iscobol.debug.remote_source_enabled to true on the remote machine. The remote runtime will look for source files in its Classpath and the iscobol.debug.code_prefix setting.

Examples

- Example of remote debugging in thin client with automatic download of source files.

    1. Start the server as follows:

```
iscserver -c server.properties
```

    2. Launch the Client as follows:

```
iscclient -J-Discobol.debug.remote_source=1 -hostname <server-ip> -d MAIN_PROGRAM
```

Content of *server.properties*:

```
iscobol.rundebug=2
iscobol.debug.remote_source_enabled=1
iscobol.debug.code_prefix=/path/to/cbl_files\n/path/to/copy_files
```

- Example of remote debugging of a character-based application (CHARVA) with automatic download of source files.

    1. Start the application as follows on the server:

```
iscrun -c runtime.properties -t MAIN_PROG
```

    2. Run the following command on the local PC:

```
iscrun -J-Discobol.debug.remote_source=1 -r <server-ip>
```

Content of *runtime.properties*:

```
iscobol.rundebug=2
iscobol.debug.remote_source_enabled=1
iscobol.debug.code_prefix=/path/to/cbl_files\n/path/to/copy_files
```

- Example of remote debugging of an EIS servlet with automatic download of source files.

    1. Add the following entries to *WEB-INF/classes/iscobol.properties*:

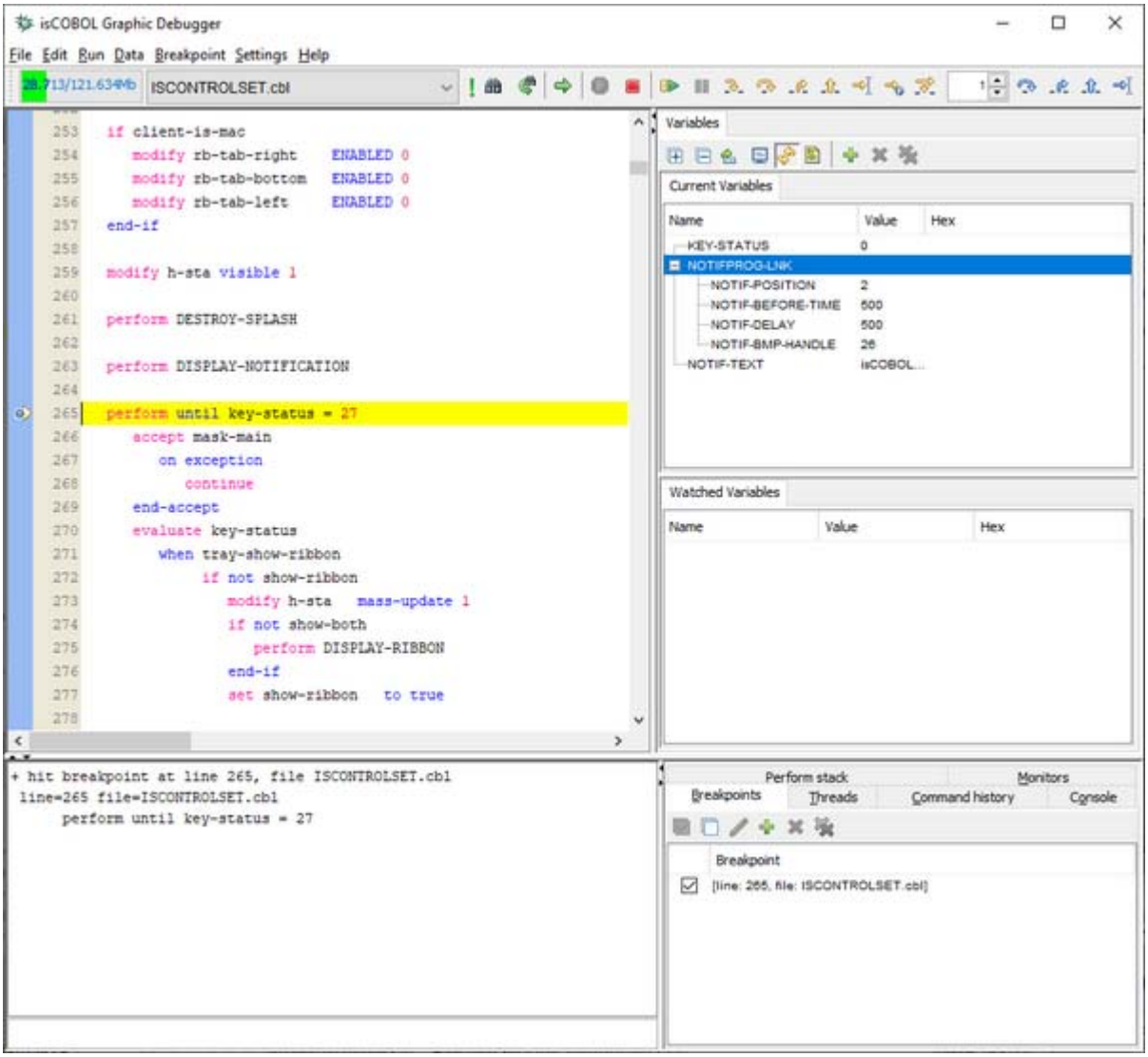
```
iscobol.rundebug=2
iscobol.debug.remote_source_enabled=1
iscobol.debug.code_prefix=/path/to/cbl_files\n/path/to/copy_files
```

    2. Navigate to the servlet's URL with your favorite browser in order to trigger the execution of the underlying COBOL program,
    3. Run the following command:

```
iscrun -J-Discobol.debug.remote_source=1 -r <servlet-container-ip>
```

The Debugger Window

The visual debugger is displayed in a dedicated window, which is divided into several areas described below:



By default the source code is shown on a white background while copybooks and nested copybooks are shown on different shades of gray. Color can be configured in Settings / Customize / Fonts And Colors.

Copybooks can be expanded and collapsed for easier reading.

Debugger command aliases and shortcuts can be configured in Settings / Customize / Commands and Settings / Customize / Shortcuts.

Settings are saved in a file named *isdebugger.properties* under the user home directory. This file is read every time the Debugger starts.

Menu Bar

On the top row of the window the Menu Bar contains links to debugger functions.

Toolbar

The toolbar contains shortcut icons and a dropdown menu for easy access to debugger functions. There are only a few that do not have a corresponding command:

Memory indicator	Shows information about memory usage. When clicked, the garbage collector feature is activated, unreferenced resources are released, and the memory heap is compacted.
Source code selector	When several source codes are loaded, it allows the user to choose which one is displayed in the Source area.
Autostep speed selector	Allows the user to change the time interval between automatic steps. The value is expressed in seconds.

After the Source Code Selector an exclamation mark icon is shown. The color of this icon provides additional information about the source code loaded by the Debugger:

Exclamation Mark Color	**Meaning**
Blue	The source code was loaded separately. It happens if you're debugging classes compiled with isCOBOL 2020 R1 or previous as well as if iscobol.debug.embedded_source (boolean) is set to false in the configuration. The blue color means that the source code is consistent with the class file, so the last modification of the main source file and the copybooks is prior to the last modification of the class file.
Green	The source code was extracted from the class file.
Red	The source code was loaded separately. It happens if you're debugging classes compiled with isCOBOL 2020 R1 or previous as well as if iscobol.debug.embedded_source (boolean) is set to false in the configuration. The red color means that the source code is not consistent with the class file, so the last modification of the main source file or one of the copybooks is more recent than the last modification of the class file. In this situation the debug experience may be affected by odd behaviors like wrong statements being highlighted by the step commands or the impossibility to set a breakpoint on a specific line. In order to resolve this issue, recompile the program.

Source Area

The main portion of the window is used to display the source code being debugged. It is interactive and allows the user to:

- Select code fragments that can be copied to the clipboard.
  Hold the left mouse button and drag the mouse to select.
  Alternatively, hold the Shift key and press the Up arrow key or the Down arrow key.
  When the desired lines are selected, right click on the area and choose 'Copy' from the context menu to copy the code to the clipboard.

- Display the value of a data item or a constant (defined as 78 level) by double clicking on it.
- See the value, size and offset of a data item or a constant (defined as 78 level) by stopping the mouse cursor on it. The tool-tip delay is configurable in Settings / Data.
- Select the next data item in the current statement or the previous data item in the current statement by pressing Tab or Shift-Tab respectively.
- Jump to a paragraph or a to a section by double clicking on its name.
- Jump to a paragraph or variable declaration by leaving the mouse pointer over the item name until it changes to a hand shape, then clicking.

Variables Area

The right portion of the window hosts the Variables Area. This area is divided into two parts:

- *Current Variables*: A list of the variables included in the current statement followed by the variables that were included in the previous statement.
- *Watched Variables:* A list of of the variables that are monitored during the debug session. In order to add a variable to this list, right click in the Variables Area and select "New watched variable" from the pop-up menu. Variables can also be added to this list through the Quick watch.

Right clicking over the Variables Area opens a pop-up menu with the following options:

Expand all	Expands the group variables shown as tree-views
Collapse all	Collapses the group variables shown as tree-views
Change value	Opens a dialog that allows you to edit the value of the selected variable
Add monitor	Creates a monitor on the selected variable
Auto Refresh	If activated, it refreshes the content of variables at each step command
Show Hex Values	Enables or disables the display of hex values in the Variables Area lists
New watched variable	Opens a dialog that allows you to add a new item to the list of Watched Variables
Remove	Removes the selected item from the list of Watched Variables
Remove All	Clears the list of Watched Variables
Copy	Copies the value to the clipboard

Output Window

The output window displays the output of the commands that you entered in the Command Area. You can scroll through the output, and copy the contents to the clipboard. The following actions are available in the context menu opened by right clicking in this area:

- Clear
- Copy
- Select All

Command Area

The command area can be used to enter debugger commands and provide input. Use [Ctrl+F8] to repeat the last command. Use [CTRL+Up Arrow] to repeat the previous command in the history and [CTRL+Down Arrow] to repeat the next one.

Information Window

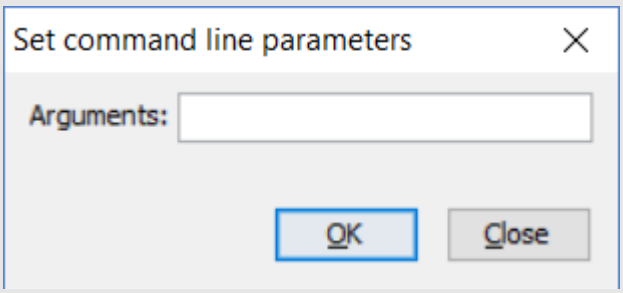
The information window contains several tabs.

Perform stack	Traces the perform stack. The content grows as you step into paragraphs and called programs and reduces as you return from them.
Monitor	Shows monitored data items and their values. These actions are permitted using the tool-bar or the context menu: • create new monitors • enable or disable monitors • modify monitors • remove monitors
Breakpoints	Shows breakpoints. These actions are permitted using the tool-bar or the context menu: • create new breakpoints • enable or disable breakpoints • modify monitors • remove breakpoints
Threads	Lists active COBOL threads. This view allows you to select which thread to debug.
Command history	Lists the commands entered so far. These actions are permitted using the tool-bar or the context menu: • execute a command again • clear the history

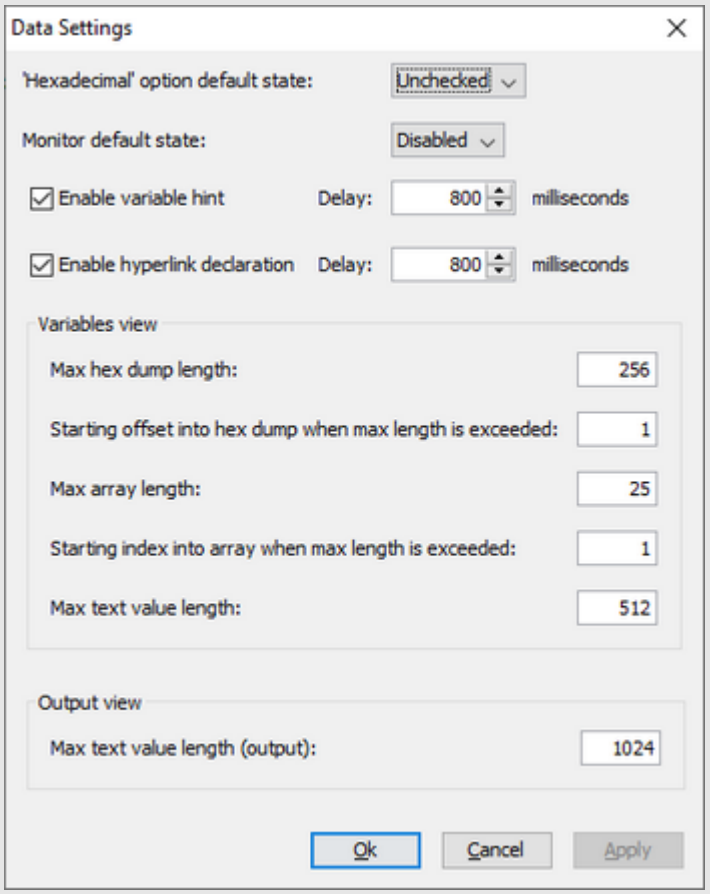
Console	Shows the console output. Both the system output and any system errors are caught in this tab. The content of the system output is shown in black. The content of a system error is shown in red. These actions are permitted using the tool-bar or the context menu: • attach or detach the console. When detached, the output is not caught by the Debugger, going to the console of the process instead (e.g. the command prompt). Note that switching between 'attach' and 'detach' during remote debugging has no effect • enable or disable the automatic activation of the Console tab when the console content changes • clear the content from the window

Debugger Functions

The following is a list of available debugger functions, how they are accessed, and their descriptions:

Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Help / About	Displays information about the current version of isCOBOL.
n/a	n/a	n/a	Run / Set command line parameters	Displays a window that allows the user to alter the parameters that were specified on the command line. 

Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Settings / Data	Displays a window that allows you to configure settings for data-items management.



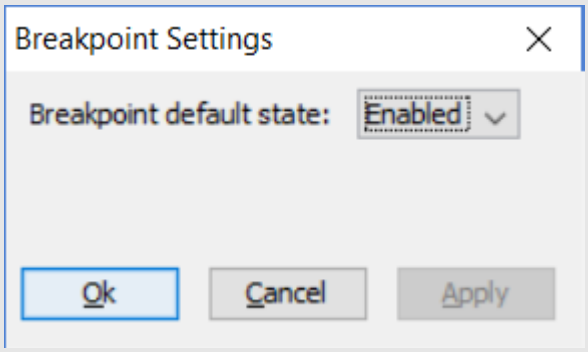
'Hexadecimal' option default state sets the default value for the *'Hexadecimal'* flag in the following dialogs:
- Data / Display
- Data / Set Monitor
- Quickwatch

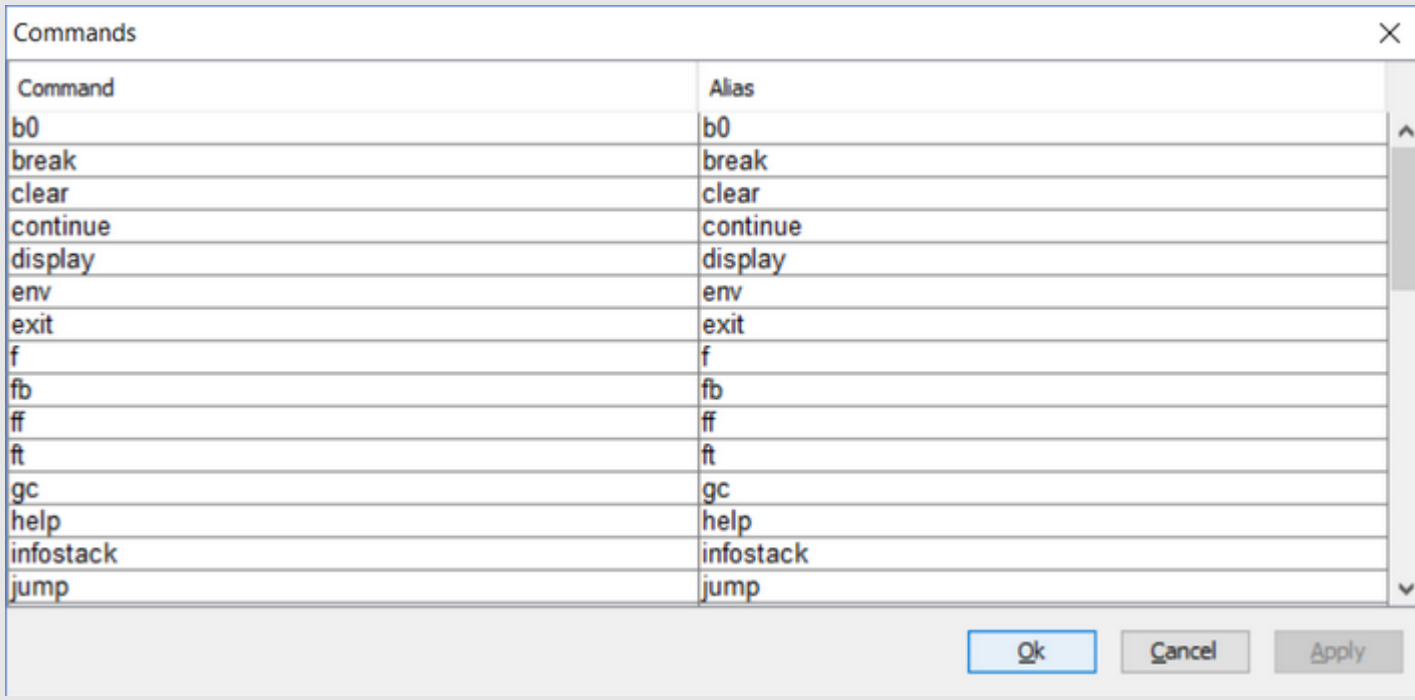
Monitor default enabled state specifies if new monitors will be enabled or not b default.

Enable variable hint enables tool-tips shown when you move the mouse pointe over a data item name. The next *Delay* field sets the delay in milliseconds for the tool-tips.

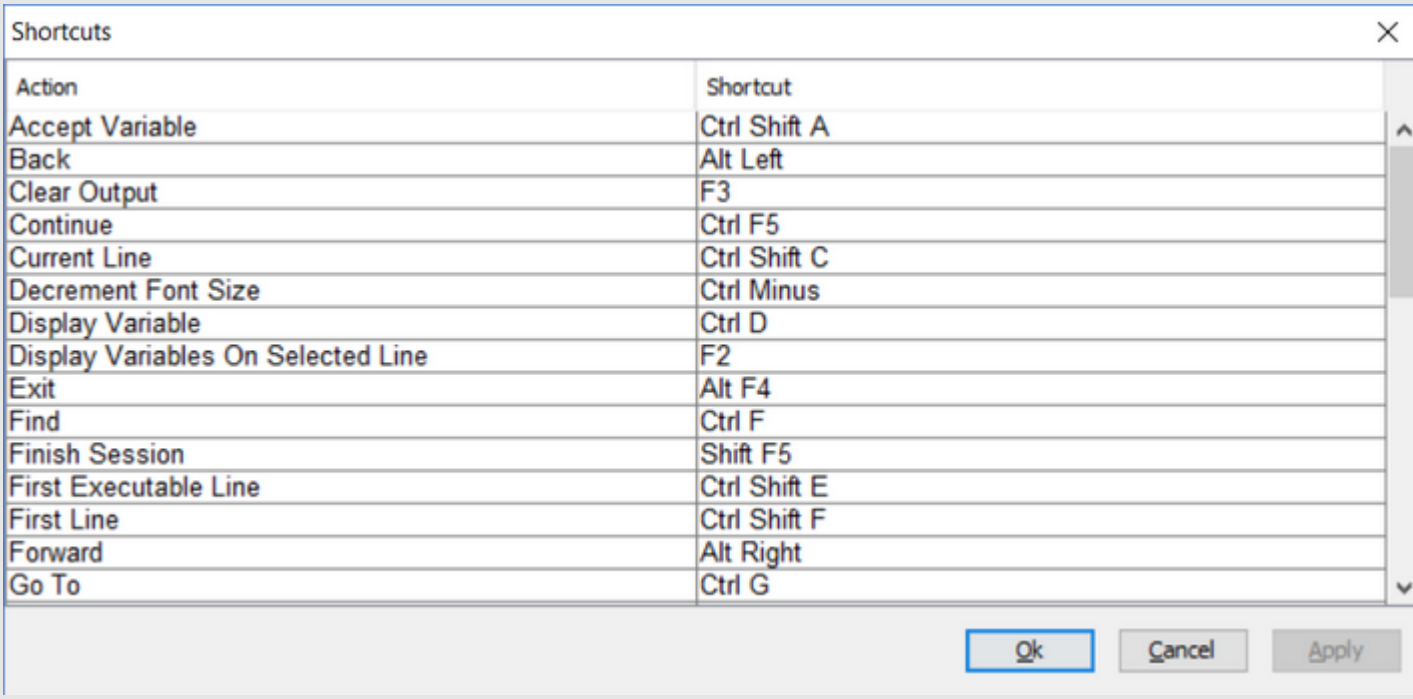
Enable hyperlink declaration enables an hyperlink effect activated when you move the mouse pointer over a data item or paragraph name. The next *Delay* field sets the delay in milliseconds for the hyperlink.

The *Variables view* area and the *Output view* areas allow you to limit the buffe used to host and display item values.

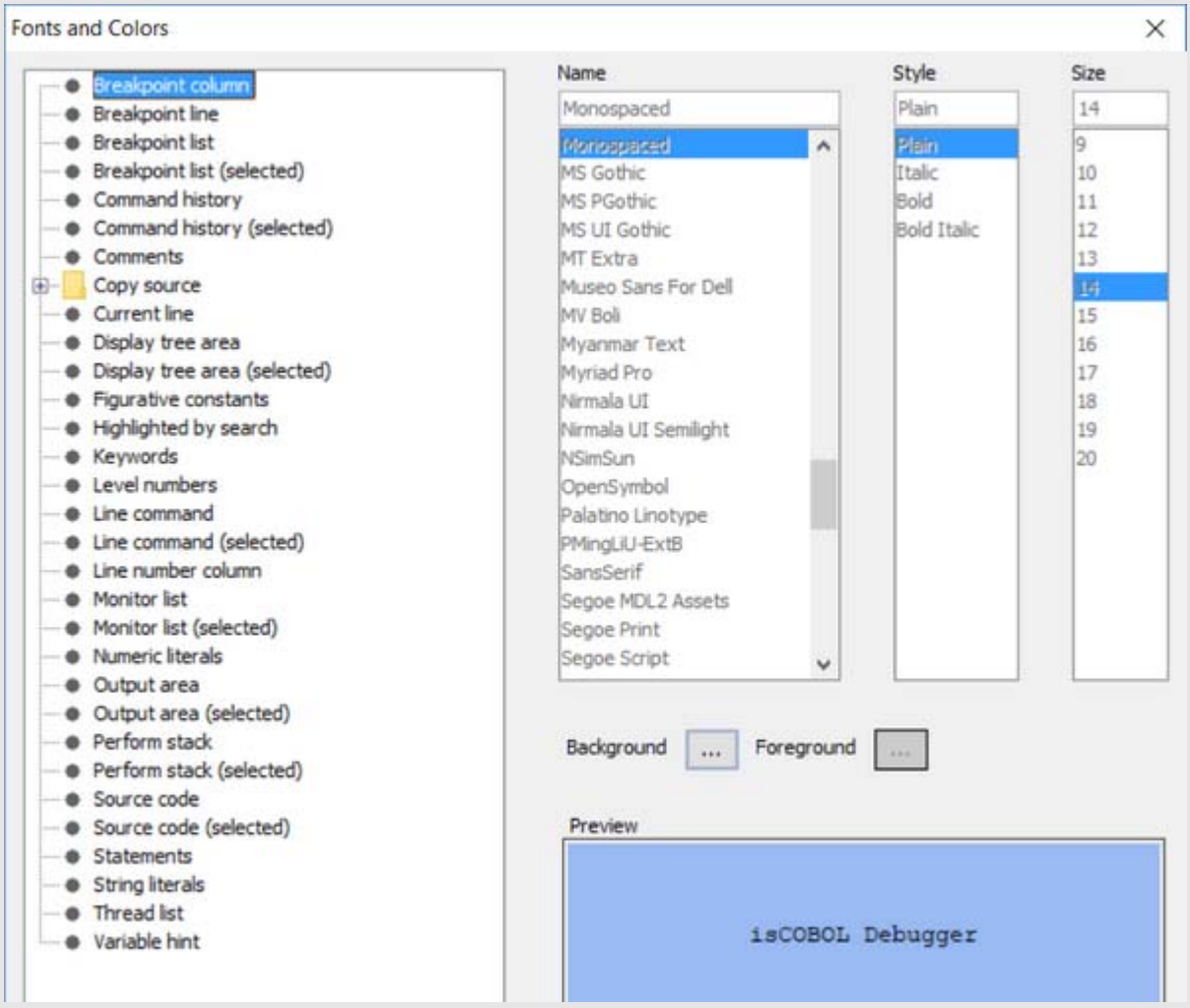
Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Settings / Breakpoint	Displays a window that allows you to configure the default state of breakpoints. 
n/a	n/a	n/a	Settings / Source / Format	Tells the Debugger which source format is used by the current program. This setting is useful only when the Debugger doesn't automatically recognize the source format and fails to color keywords and comments.
			Settings / Source / Expand copy books when loading source	Tells the Debugger to automatically expand the copy books included in the program source code.
n/a	n/a	[Ctrl++] [Ctrl+-]	Settings / Font size	Sets the size of the font used to display the source code.
n/a	n/a	n/a	Settings / Customize / Commands	Displays a window that allows the user to create an alias for every Debugger command.

Command	Shortest Form	Key	Menu	Description
				
n/a	n/a	n/a	Settings / Customize / Shortcuts	Displays a window that allows the user to customize the keyboard shortuct fo every Debugger command.

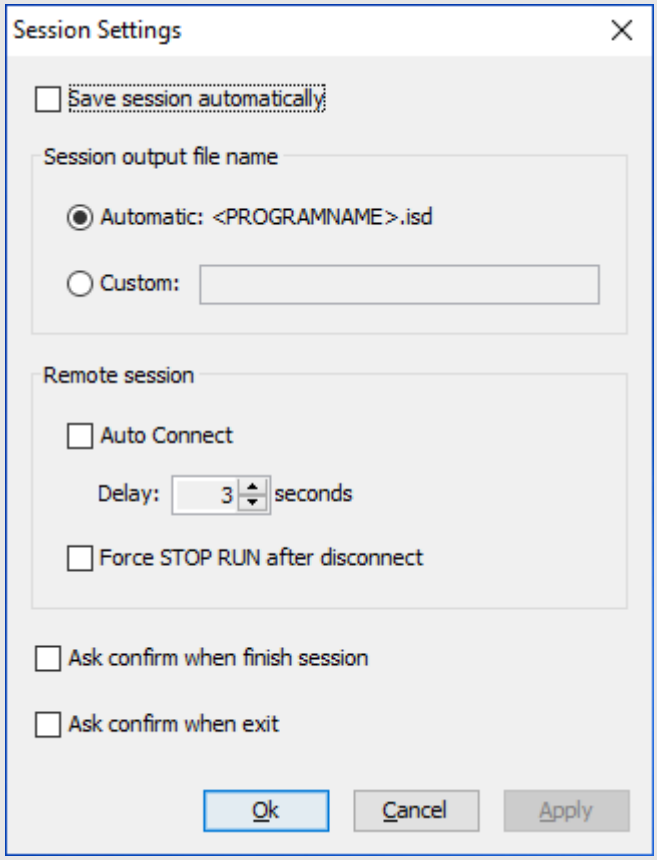
Command	Shortest Form	Key	Menu	Description



Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Settings / Customize / Fonts And Colors	Displays a window that allows the user to customize the appearance of the various elements of the debugger window.



Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Settings / Session	Displays a window that allows the user to customize the debugger session.



Save session automatically enables the automatic save of Debugger sessions o exit.

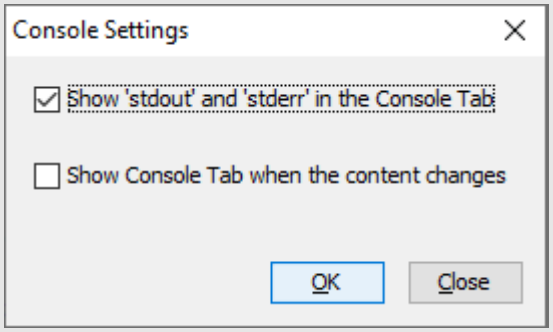
Session output file name allows you to specify a custom name for the file wher the Debugger session is saved.

Auto Connect allows you to specify how many seconds the Debugger should wait before connecting to a remote runtime.

When *Force STOP RUN after disconnect* is checked, the debugged program will perform a STOP RUN after the remote debugger is either disconnected or closed. If the program is accepting user input, the STOP RUN will occur as soo as the ACCEPT is interrupted.

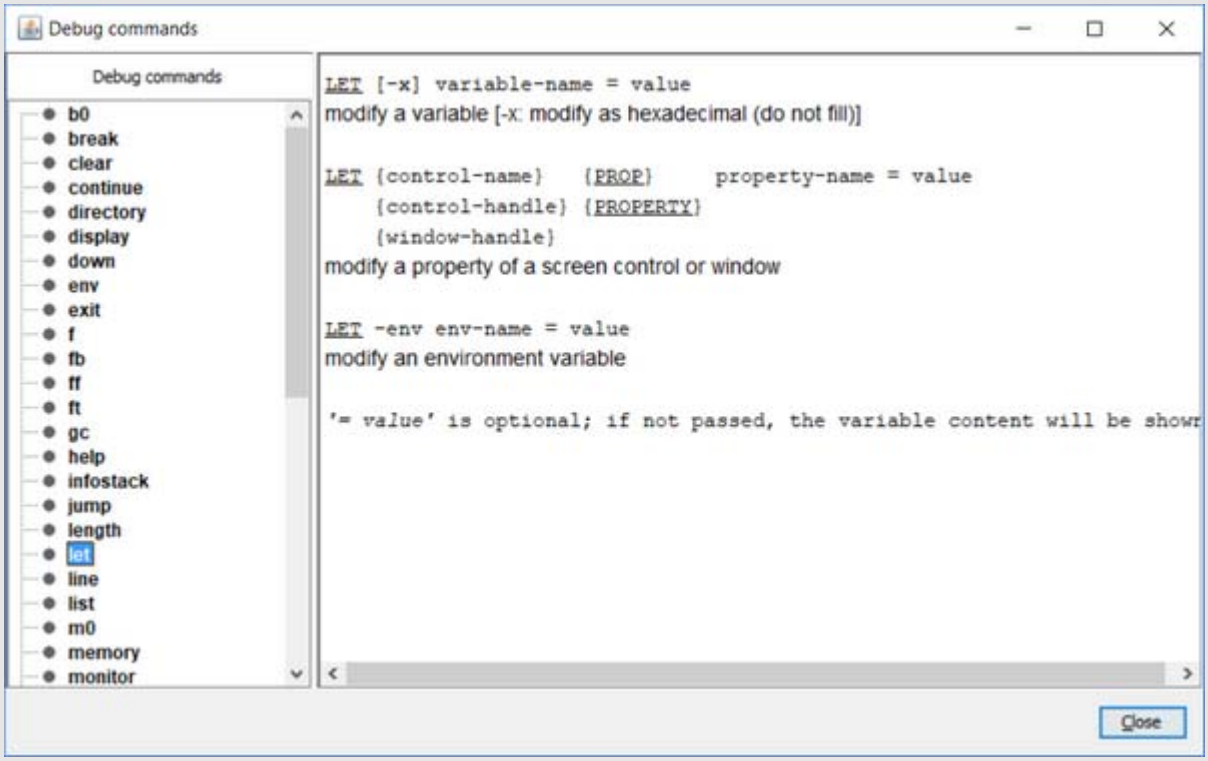
Ask confirm... options allows you to enable a prompt message to be shown when the user chooses to terminate the session or exit from Debugger.

Command	Shortest Form	Key	Menu	Description
n/a	n/a	n/a	Settings / Console	Displays a window that allows the user to customize the Console tab.

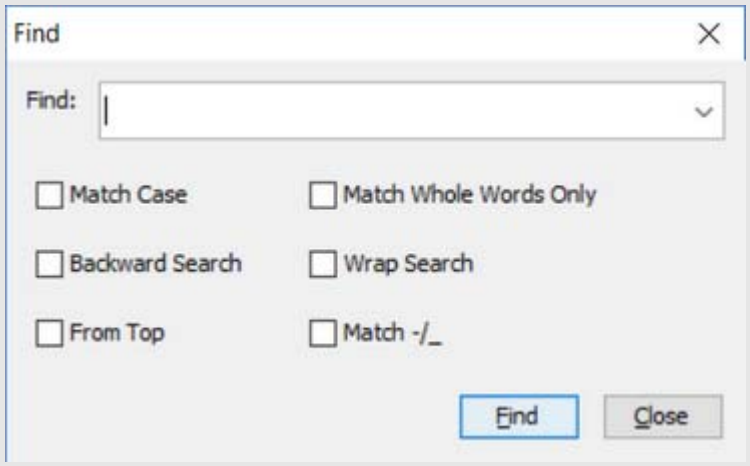


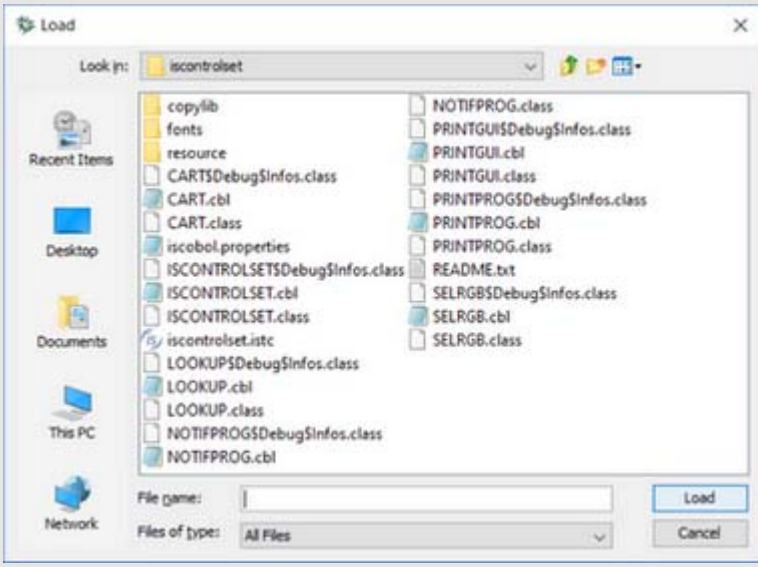
The choices made here are reflected in the tool-bar buttons of the Console tab.

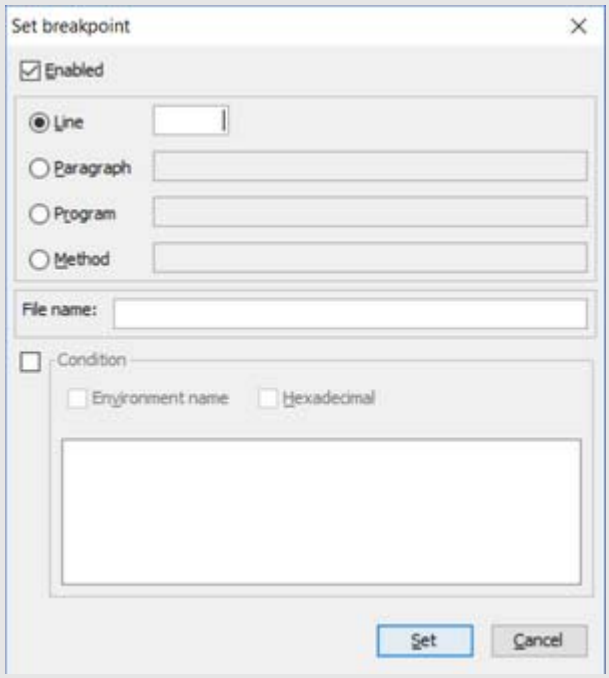
n/a	n/a	[F1]	Help / Commands	Displays a window containing a list of debugger commands and their use.



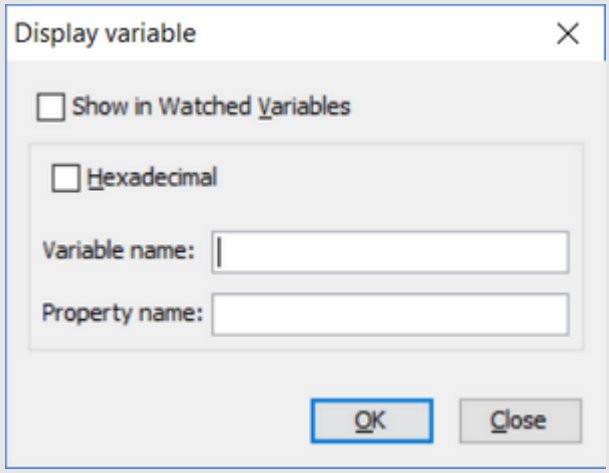
n/a	n/a	[F2]	Data / Display variables on selected line	Displays all variables that appear in the selected source line along with their current value.
n/a	n/a	[F3]	Edit / Clear output	Clears the Output Window.

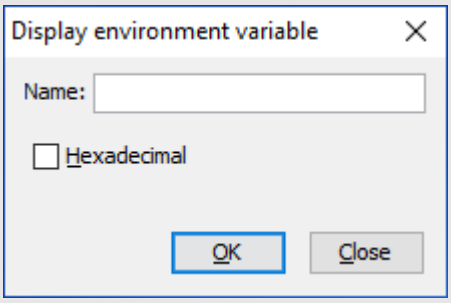
Command	Shortest Form	Key	Menu	Description
n/a	n/a	[F4]	Breakpoints / Toggle at current line	Toggles a breakpoint at the current line of the current source code.
n/a	n/a	[F9]	Run / Go to cursor line	Starts or continues the program execution until the line where the cursor is located is reached.
n/a	n/a	[Ctrl+F8]	Edit / Last command	Repeats the last command entered in the Command Area. The command is no immediately executed, so the user can change it before executing.
n/a	n/a	[Ctrl+F]	Edit / Find	Displays a window that searches text in the current source code. 
n/a	n/a	n/a	Edit / Expand all copy books	Expands all the copybooks in the current source file
n/a	n/a	n/a	Edit / Collapse all copy books	Collapses all the copybooks in the current source file
n/a	n/a	[Ctrl+G]	Edit / Go To	Displays a window that jumps to a specific part of one of the currently loaded programs.

Command	Shortest Form	Key	Menu	Description
n/a	n/a	[F12]	Edit / Go to declaration	Having a variable selected in any part of the source, moves the cursor to the variable declaration in Data Division.
n/a	n/a	[Alt+Left]	Edit / Back	Moves the cursor to the previous occurrence of the selected variable in the source.
n/a	n/a	[Alt+Right]	Edit / Forward	Moves the cursor to the next occurrence of the selected variable in the source
n/a	n/a	[F3]	Edit / Clear output	Clears the content of the output window.
n/a	n/a	[Ctrl+L]	File / Load file	Loads a source file. The purpose of loading a source file is to set breakpoints i that file before executing it. 
n/a	n/a	[Ctrl+U]	File / Unload current file	Releases a previously loaded source file.
b0	n/a	n/a	n/a	Usage: *b0 [{ -d | -e }] ProgramName* Sets a breakpoint at the beginning of the program *ProgramName.* If the-d option is used, the breakpoint is disabled. If the-e option is used, the breakpoint is enabled.

Command	Shortest Form	Key	Menu	Description
break	br	[Ctrl+B]	Breakpoints / Set	Usage: *break* Displays a window that allows the user to set breakpoints. 
		n/a	n/a	Usage: *break [{ -d |-e }] { LineNumber | ParagraphName } [SourceCode] [when WhenConditions]* Sets a breakpoint. When "`-d`" is specified, the breakpoint is disabled. When "`-d`" is specified, the breakpoint is enabled. *LineNumber* is the line number to which the breakpoint refers. That line must contain a statement. If a statement is split between several lines, the breakpoint can only be set at the first line. *ParagraphName* is the name of a paragraph. The breakpoint will refer to its first line. *SourceCode* is the optional name of the source code to which *LineNumber* and *ParagraphName* refer. If *SourceCode* is not specified, the current source code is implied. *WhenConditions* is: *[-x][-env] VariableName Operator Value [LogicalOperator VariableName Operator Value] ...* Where: *VariableName* is the data item to monitor. *Operator* can be =, !=, <, >, <=, >=. *LogicalOperator* can be either && or II. *Value* is the value to be tested. if -x is used, the value is hexadecimal if -env is used, the variable is searched for amongst configuration properties
		n/a	View / Breakpoints	Usage: *break -l* Activates the Breakpoint view in the Information Window. All breakpoints currently set are listed.

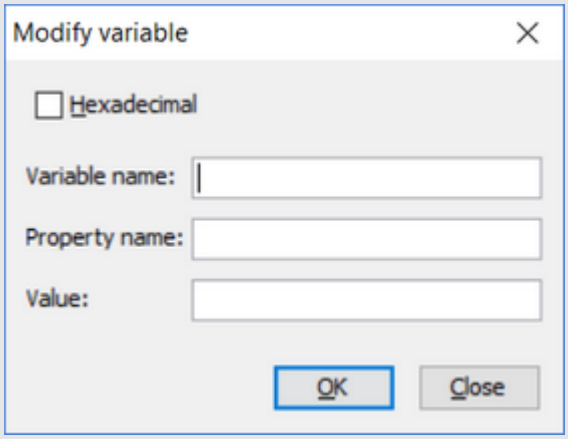
Command	Shortest Form	Key	Menu	Description
clear	cl	n/a	n/a	Usage: *clear { LineNumber | ParagraphName } [SourceCode]* Clears a breakpoint. *LineNumber* is the line number where the breakpoint is set. *ParagraphName* is the name of the paragraph where a breakpoint is set. *SourceCode* is the optional name of the source code that *LineNumber* and *ParagraphName* refer to. If *SourceCode* is not specified, the current source cod is implied.
		n/a	Breakpoints / Clear all	Usage: *clear -l* Clears all breakpoints.
continue	co	[Ctrl+F5]	Run / Continue	Usage: *continue* Starts or continues program execution.
directory	dir	n/a	n/a	This is a deprecated command. It's useful only if you're debugging programs compiled by isCOBOL 2020 R1 or previous. Usage: *directory [DirectoryName]* If *DirectoryName* is omitted, then the value of iscobol.debug.code_prefix is shown. Otherwise, the directory specified by *DirectoryName* is appended to th value of iscobol.debug.code_prefix.

Command	Shortest Form	Key	Menu	Description
display	dis	[Ctrl+D]	Data / Display	Usage: *display* Opens a window that displays the value of a variable. Refer to the usages below for details. 
		n/a	n/a	Usage: *display [-x] [-tree] VariableName* Displays the value of a data item. When "-x" is specified, the hexadecimal value is shown. When "-tree" is specified, a new tab is added to the information window. It will show the data item and all its sub-levels in a hierarchical structure. It can be updated or removed by right-clicking in the tab to display a contextual menu. When "-x" and "-tree" are specified in the same command, their effects are combined. *VariableName* is the data item whose value will be displayed.
		n/a	n/a	Usage: *display [-x] ControlHandle [property | prop] PropertyName* Displays the current value of a control property. When "-x" is specified, the hexadecimal value is shown. *ControlHandle* must refer to a valid handle. *PropertyName* is a the name of a property of *ControlHandle*.
display -classversion	dis -classversion	n/a	Run / Display isCOBOL version	Usage: *display -classversion* Prints the version of the Compiler that produced the current class.

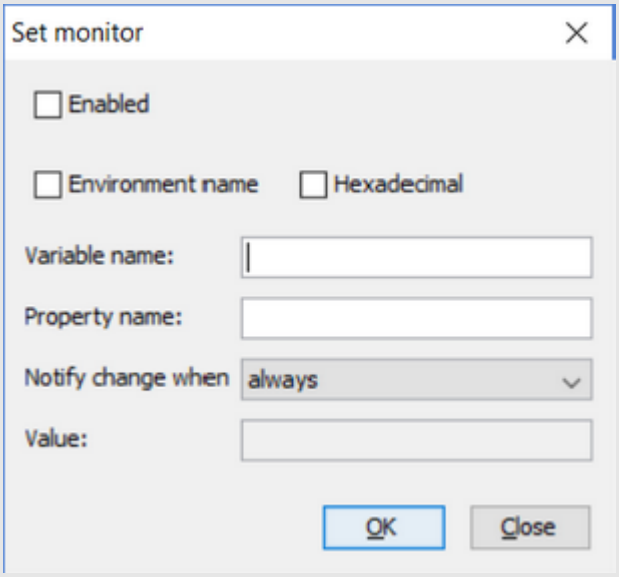
Command	Shortest Form	Key	Menu	Description
display -env	dis -env	n/a	Data / Display Environment variable	Usage: *display [-x] -env VariableName* Displays the value of an environment variable. When "-x" is specified, the hexadecimal value is shown. *VariableName* is the name of the environment variable to be displayed. When activated by menu, the following dialog is shown: 
down	do	n/a	n/a	Usage: *down* Shows the next lower stack frame.
env	en	n/a	Run / Display environment variable	Usage: *env* Opens a window that allows the user to enter the name of the environment variable to be displayed.
		n/a	n/a	Usage: *env VariableName* Displays the value of an environment variable. *VariableName* is the name of the environment variable to be displayed.
exit	ex	[Alt+F4]	File / Exit	Usage: *exit* Terminates the debugging session and exits.
f	n/a	n/a	Edit / Repeat find	Usage: *f* Repeats the last search, with the same options
fb	n/a	n/a	Edit / Find backwards	Usage: *fb SearchText* Searches backwards for specific text. *SearchText* is the text to be searched for.

Command	Shortest Form	Key	Menu	Description
ff	n/a	n/a	Edit / Find forwards	Usage: *ff SearchText* Searches forward for specific text. *SearchText* is the text to be searched for.
ft	n/a	n/a	Edit / Find from top	Usage: *ft SearchText* Searches for specific text from the beginning of the source. *SearchText* is the text to be searched for.
gc	g	n/a	n/a	Usage: *gc* Forces the garbage collector to release unreferenced resources and compact the memory heap.
help	h	n/a	n/a	Usage: *help* Lists all the available debugger commands.
		n/a	n/a	Usage: *help DebuggerCommand* Displays the usage of a specific debugger command. *DebuggerCommand* is the command to be searched for.
infostack	i	n/a	View / Perform stack	Activates the "Perform stack" tab in the Information window.

Command	Shortest Form	Key	Menu	Description
jump Note - this is supported only in programs compiled with -dx option j		[Ctrl+J]	Run / Jump to	Usage: *jump* Opens a window that allows the user to jump to a specific line by skipping the code between the current line and the destination line.
		n/a	Run / Jump to selected line	Usage: *jump line-number [filename]* Jump to a specific line by skipping the code between the current line and the destination line. **Note:** Jumping to lines that are inside blocks is not allowed. In this case the Debugger jumps to the beginning of the block.
		n/a	n/a	Usage: *jump paragraph-name* Jump to a specific paragraph by skipping the code between the current line and the destination line.
		n/a	Run / Jump out paragraph	Usage: *jump -outpar* Jump out of the current paragraph skipping all the remaining statements in the paragraph.
		n/a	Run / Jump out program	Usage: *jump -outprog* Jump out of the current program skipping all the remaining statements in the program.
		n/a	Run / Jump next statement	Usage: *jump -next* Jump to the next statement skipping the current one.
length	len	n/a	Data / Length...	Usage: *length variable-name* Displays the lenght in bytes of a data item.

Command	Shortest Form	Key	Menu	Description
let	le	[Ctrl+Shift+A]	Data / Accept	Usage: *let* Opens a window that allows the user to change the value of a variable. Refer t the usages below for details. 
		n/a	n/a	Usage: *let [-x] VariableName [=VariableValue]* Changes the value of a data item. When "`-x`" is specified, a hexadecimal value must be entered. *VariableName* is the data item whose value will be changed. *VariableValue* is the value that will be set to *VariableName*. If omitted, the current variable content is shown and you're allowed to change it.
		n/a	n/a	Usage: *let ControlHandle { property | prop } PropertyName [=PropertyValue]* Changes the current value of a control property. *ControlHandle* must refer to a valid handle. *PropertyName* is a the name of a property of *ControlHandle*. *PropertyValue* is the value that will be set to *PropertyName*. If omitted, the current property value is shown and you're allowed to change it.
		n/a	Run / Accept environment variable	Usage: *let -env VariableName [=VariableValue]* Changes the value of a configuration property. *VariableName* is the data item whose value will be changed. *VariableValue* is the value that will be set to *VariableName*. If omitted, the current property value is shown and you're allowed to change it.
line	lin	n/a	n/a	Shows information about the current line.
list	lis	n/a	n/a	Shows some lines of code, starting at the current line.

Command	Shortest Form	Key	Menu	Description
m0	n/a	n/a	n/a	Usage: *m0 [{ -d | -e }][classname { . | :> | :: }][methodname]([signature])* Sets a breakpoint at the first executable line of *classename.methodname*. If *classname* is not specified, the breakpoint is set on the current debugged class. If *signature* is not specified and there is only a method named *methodname*, the breakpoint is set on that method. *signature* is a comma separated list of class names or primitive types names, e.g. (java.lang.String,int,java.awt.Rectangle,boolean) If the -d option is used, the breakpoint is disabled. If the -e option is used, the breakpoint is enabled.
memory	me	n/a	n/a	Shows information about memory usage.

Command	Shortest Form	Key	Menu	Description
monitor	mo	[Ctrl+M]	Data / Set monitor	Usage: *monitor* Opens a window that allows the user to enter the parameters needed to set a new monitor  Note - the content of the Value field is trimmed, unless you delimit it by quote
		n/a	n/a	Usage: *monitor [-d] [-e] [-x] VariableName [when Operator Value | always | never* Monitors a data item. When its value changes or matches a condition, the execution of the program is suspended and the debugger is activated. *VariableName* is the data item to monitor. *Operator* can be =, !=, <, >, <=, >=. *Value* is the value to be tested. If you need to include leading or trailing space in the value, delimit it by quotes. When "-d" is specified, the monitor is disabled and its value in the Information Window is not updated. When "-e" is specified, the monitor is enabled and its value in the Information Window is updated. When "-x" is specified, the value is hexadecimal. When the "always" phrase is specified, the debugger is activated each time th value changes. When the "never" phrase is specified, the debugger is never activated, but the value in the Information Window is always updated.

Command	Shortest Form	Key	Menu	Description
				Usage: *monitor [-d][-e] ControlHandle [property | prop] PropertyName [when Operator PropertyValue | always | never]* Monitors a property of a control. When its value changes or matches a condition, the execution of the program is suspended and the debugger is activated. *ControlHandle* must refer to a valid handle. *PropertyName* is a the name of the property of ControlHandle to monitor. *Operator* can be =, !=, <, >, <=, >=. *PropertyValue* is the value to be tested. When "-d" is specified, the monitor is disabled and its value in the Information Window is not updated. When "-e" is specified, the monitor is enabled and its value in the Information Window is updated. When the "always" phrase is specified, the debugger is activated each time the value changes. When the "never" phrase is specified, the debugger is never activated, but the value in the Information Window is always updated.
		n/a	n/a	Usage: *monitor [-d][-e] -env VariableName* Monitors an environment variable. When "-d" is specified, the monitor is disabled and its value in the Information Window is not updated. When "-e" is specified, the monitor is enabled and its value in the Information Window is updated. *VariableName* is the name of the environment variable to monitor.
		n/a	View / Monitors	Usage: *monitor -l* Activates the Monitors view in the Information Window. All monitors currently set are listed.
next	n	[Shift+F7]	Run / Step over	Usage: *next* Executes the current statement. If it is a PERFORM statement, it is entirely executed.
offset	of	n/a	Data / Offset...	Usage: *offset variable-name* Displays the offset of a data item.
outpar	outpa	[Alt+Shift+F7]	Run / Step out paragraph	Usage: *outpar* Continues execution until current paragraph exits.

Command	Shortest Form	Key	Menu	Description
outprog	outpr	[Alt+Shift+F8]	Run / Step out current program	Usage: *outprog* Continues execution until current program exits.
pause	p	n/a	Run / Pause	Usage: *pause* Suspends the program execution.
prog	prog	[Alt+F9]	Run / Run to next program	Usage: *prog* Continues execution until the runtime enters in the next program compiled in debug mode.
quit	q	[Shift+F5]	Run / Finish session	Usage: *quit* Stops the execution of the program. The debugging session is still valid and th program can be restarted with the run command.
readsession	re	n/a	File / Load debugger session	Usage: *readsession [FileName]* Loads monitors and breakpoints from a previously saved debugging session. FileName is the name of the file that contains the debugger configuration. If i is not specified, '`ISCONTROLSET.isd`' is implied.
run	ru	[Ctrl+F6]	Run / Start session	Usage: *run* Starts the execution of the program. No COBOL statements are executed.
step	s	[F7]	Run / Step into	Usage: *step [n]* Executes the current statement. If it is a `PERFORM` statement, the first statement of the paragraph or session that it refers to becomes current. If *n* is specified and it's greater than 1, the step command is automatically repeated times.
stoff	n/a	[Ctrl+F4]	Run / Stop autostep	Usage: *stoff* Deactivates the autostep function.
ston	n/a	[Ctrl+F3]	Run / Start autostep	Activates the autostep function. Statements are automatically executed at regular intervals. The function can b changed with the selector on right side of the toolbar.
thread	th	n/a	Run / ThreadName	Usage: *thread ThreadName* Activates a specific thread. *ThreadName* is the name of the thread to activate.
		n/a	View / Threads	Usage: *thread -l* Activates the Threads view in the Information Window. All monitors currently set are listed.

Command	Shortest Form	Key	Menu	Description
to	to	[Ctrl+F9]	Run / Continue to line number	Usage: *to LineNumber [SourceCode]* Starts or continues the program execution until a certain line is reached. *LineNumber* is the line to reach. *SourceCode* is the optional name of the source code to which *LineNumber* refers. If it is not specified, the current source code is implied.
troff	trof	[Shift+F4]	Trace Off	Usage: *troff* Tracing is suspended.
tron	n/a	[Shift+F3]	Trace On	Usage: *tron* Tracing is activated. A trace line is appended each time a paragraph starts or ends and each time a program starts or ends. Information is stored in a file called debugger.log, created in the current directory.
unmonitor	u	n/a	n/a	Usage: *unmonitor VariableName* Removes the monitor on a data item. *VariableName* is a monitored data item.
		n/a	n/a	Usage: *unmonitor [-env] VariableName* Removes a monitor on an environment variable. *VariableName* is a monitored environment variable .
		n/a	Data / Clear all monitors	Usage: *unmonitor -a* Clears all monitors.
up	up	n/a	n/a	Usage: *up* Shows the higher stack frame.
w0	n/a	n/a	Edit / First executable line	Usage: *w0* Moves the cursor to the first executable line.
w@	n/a	n/a	Edit / Current line	Usage: *w@* Moves the cursor to the current line.
wb	n/a	[Ctrl+Home]	Edit / Last line	Usage: *wb* Moves the cursor to the last line of the current source.
wt	n/a	[Ctrl+End]	Edit / First line	Usage: *wt* Moves the cursor to the first line of the current source.

Command	Shortest Form	Key	Menu	Description
writesession	w	n/a	File / Save debugger session	Usage: *writesession [FileName]* Saves monitors and breakpoints to a file. *FileName* is the name of the file that contains the debugger configuration. If it i not specified, 'ISCONTROLSET.isd' is implied.

Enter Debugger

If a program compiled in debug mode is performing ACCEPT of user input on a graphical window, if you press Pause/Break on the keyboard, you will enter Debugger at the next ACCEPT interruption. For example, if you want to debug what happens when you click on a specific push-button of your window,

1. start the Debugger
2. issue the run command
3. issue the continue command
4. wait for the window with the push-button to appear
5. press Pause/Break on the keyboard
6. click on the push-button

On some keyboards the Pause/Break key is not available. In order to have the same feature associated to another key, set the exception value of that key to 65535. For example, if you want to enter debugger using F6, start the Debugger as follows:

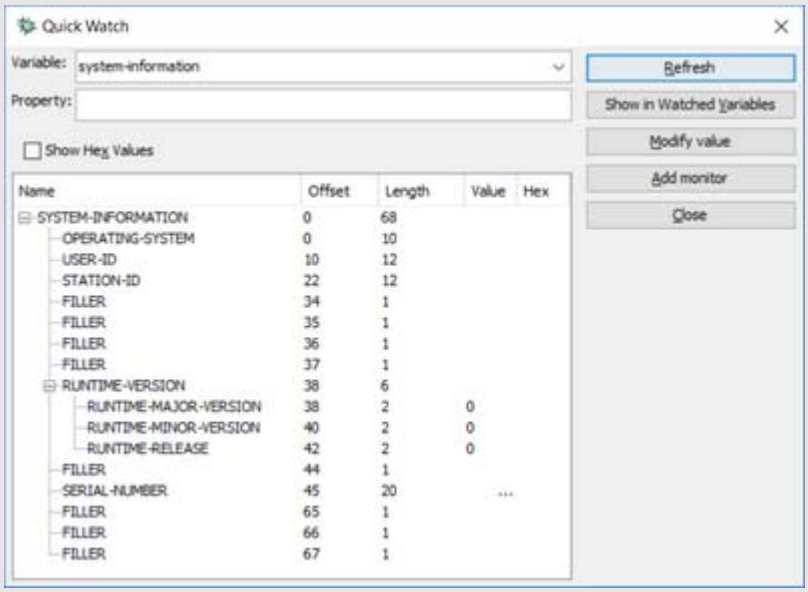
```
iscrun -d -Discobol.key.f6=exception=65535 MYPROGRAM
```

Pop-up menu

When you right click in the Source Area, a pop-up menu appears and provides the following functions:

Copy	Copies the selected line(s) in the system clipboard. The same result is achieved by pressing Ctrl+C
Current line	Same function as Edit / Current line
Go to	Same function as Edit / Go To
Continue	Same function as Run / Continue
Pause	Same function as Run / Pause
Step into	Same function as Run / Step into
Step over	Same function as Run / Step over
Step out paragraph	Same function as Run / Step out paragraph
Step out program	Same function as Run / Step out current program
Run to selected line	Same function as Run / Go to cursor line

Run to next program	Same function as Run / Run to next program
Jump out paragraph	Same function as Run / Jump out paragraph
Jump out program	Same function as Run / Jump out program
Jump to selected line	Same function as Run / Jump to selected line
Toggle breakpoint	Same function as Breakpoints / Toggle at current line
Display variables on selected line	Same function as Data / Display variables on selected line
Quick watch	Opens a dialog that allows you to handle the selected data item



Debugger Properties

The list of configuration properties that affect the Debugger behavior can be found at Debugger Configuration.

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

Character-based Debugger

isCOBOL provides a character-based version of the Visual Debugger to be used on systems where the UI is not available. The character-based Debugger is started using the isdbg command. This command has the following syntaxes:

- Local debug:

```
isdbg [-opt1 ... -optN] program-name [arg1 ... argN]
```

- Remote debug:

```
isdbg -r [hostname [port]]
```

Using one of the above commands the debugger console starts and listens for input:

```
isdb>
```

Input the desired command and press Enter to confirm. The command output is displayed on the console.

ISDBG Commands

Command	Description
`b0`	Usage: b0 *prog-name*
	set a breakpoint at the beginning of a given program
`break`	Usage: break *line-number|paragraph-name*
	set a breakpoint at a given line or paragraph
	Usage: break -l
	list breackpoints
`clear`	Usage: clear *line-number|paragraph-name*
	remove a breakpoint at a given line or paragraph
	Usage: clear -a
	remove all breakpoints
`continue`	Usage: continue
	continue execution until the next breakpoint

Command	Description
`directory`	Usage: directory *dir-name*
	Add a given directory to the debug code_prefix
	Usage: directory
	Shows the current debug code_prefix
`display`	Usage: display variable-name
	display the current value of a variable in ascii or decimal
	Usage: display -x variable-name
	display the current value of a variable in hex
`down`	Usage: down
	View the next lower stack frame
`exit`	Usage: exit
	exit debug
`f`	Usage: f
	repeat find
`fb`	Usage: fb *text*
	find text backwards
`ff`	Usage: ff *text*
	find text forwards
`ft`	Usage: ft *text*
	find text from top
`gc`	Usage: gc
	force garbage collector
`help`	Usage: help
	show help
`infostack`	Usage: infostack
	display stack information
`jump`	Usage: jump *line-number|paragraph-name*
	jump to a given line or paragraph

Command	Description
`let`	Usage: let *variable-name=value* assign new value to a variable
`line`	Usage: line display the current line of source code
`list`	Usage: list display the source code
`memory`	Usage: memory print memory information
`monitor`	Usage: monitor *variable-name* set a monitor on a given variable Usage: monitor -l list monitors
`next`	Usage: next step one line (step over CALL and PERFORM statements)
`outpar`	Usage: outpar step out of the current paragraph
`outprog`	Usage: outprog step out of the current program and return to the caller
`pause`	Usage: pause pause execution
`quit`	Usage: quit stop execution
`run`	Usage: run start execution
`step`	Usage: step execute the next statement Usage: step *n* execute the next *n* statements

Command	Description
`stoff`	Usage: stoff
	stop autostep
`ston`	Usage: ston
	start autostep
`thread`	Usage: thread *thread-name*
	choose the thread to debug
	Usage: thread -l
	list threads
`to`	Usage: to *line-number*
	continue execution until the given line number is reached
`troff`	Usage: troff
	stop tracing program execution
`tron`	Usage: tron *tracelevel log-filename*
	start tracing program execution on a text file. See iscobol.tracelevel for possible *tracelevel* values.
`unmonitor`	Usage: unmonitor *variable-name*
	clear the monitor on a specified variable
`up`	Usage: up
	View the next higher stack frame
`w0`	Usage: w0
	go to first executable line
`w@`	Usage: w@
	show the current line
`wb`	Usage: wb
	show last line of source code
`wt`	Usage: wt
	show first line of source code

Chapter 3

RemoteCompiler

Overview

isCOBOL Evolve offers the ability to compile programs remotely. This feature is particularly useful when you need to precompile the source code and the precompiler is available only on a server machine.

The isCOBOL RemoteCompiler consists of a server listener that receives source files from the clients, precompiles and compiles them locally on the server, and eventually sends the resulting classes and translated source files back to the clients.

Getting Started

The setup of a RemoteCompiler 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 RemoteCompiler. For best results and performance, install the latest JDK version available for your platform. isCOBOL is certified to work correctly with both Oracle JDK and OpenJDK from version 8 to version 17.

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

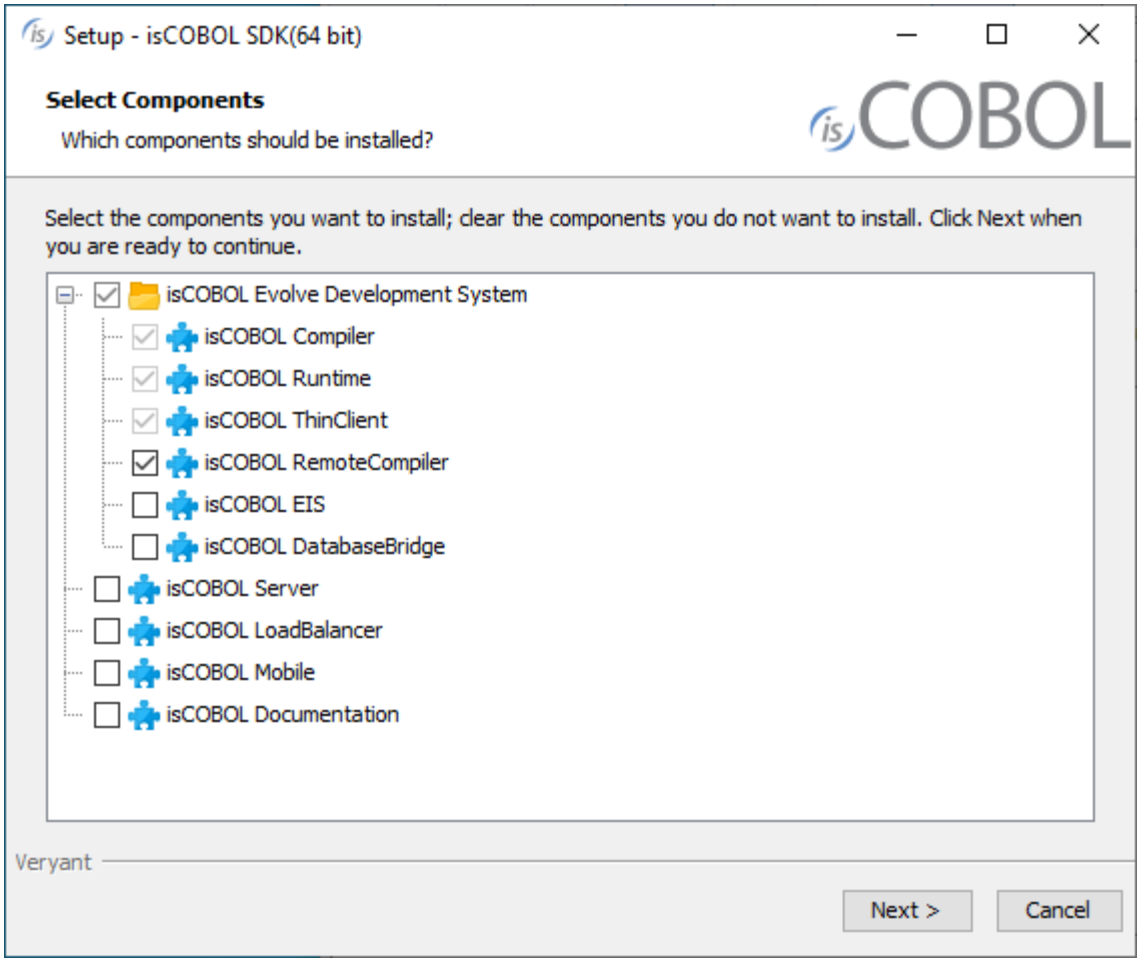
Windows

1. If you haven't already done so, Download and install the Java Development Kit (JDK).
2. Go to "https://support.veryant.com".
3. Sign in with your User ID and Password.

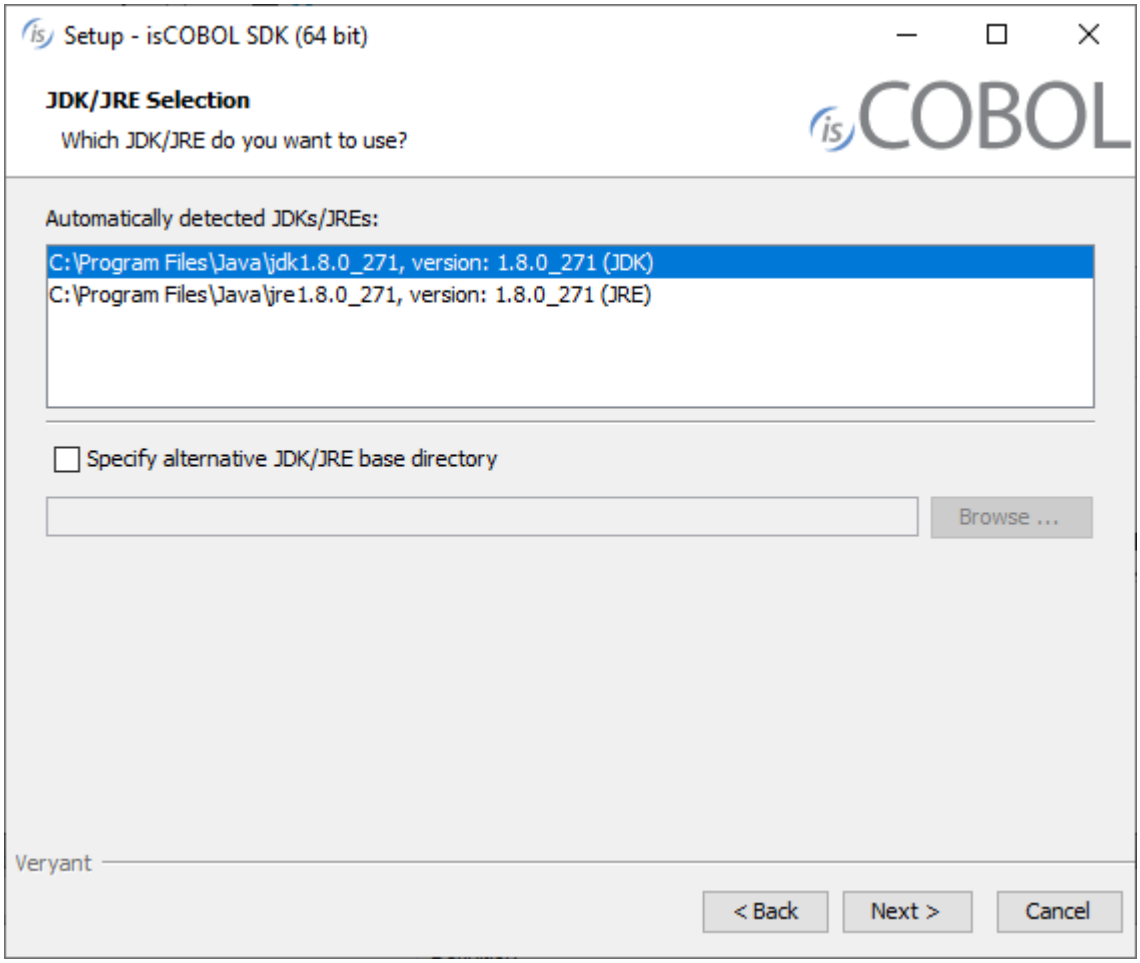
4. Click on the "Download Software" link.
5. Scroll down to the list of files for Windows x64 64-bit or Windows x86 32-bit. Select isCOBOL_2023_R1_*n*_Windows.*arc*.msi, where *n* is the build number and *arc* is the system architecture.
6. Run the downloaded installer to install the files.

Note - If your Windows has the option "Run as Administrator", you should run the setup with that option, otherwise the setting of environment variables might silently fail. Environment variables setting is not necessary if you work from the isCOBOL Shell (explained later).

7. Select "isCOBOL Compiler and Runtime Environment" and "isCOBOL RemoteCompiler" from the list of products when prompted.

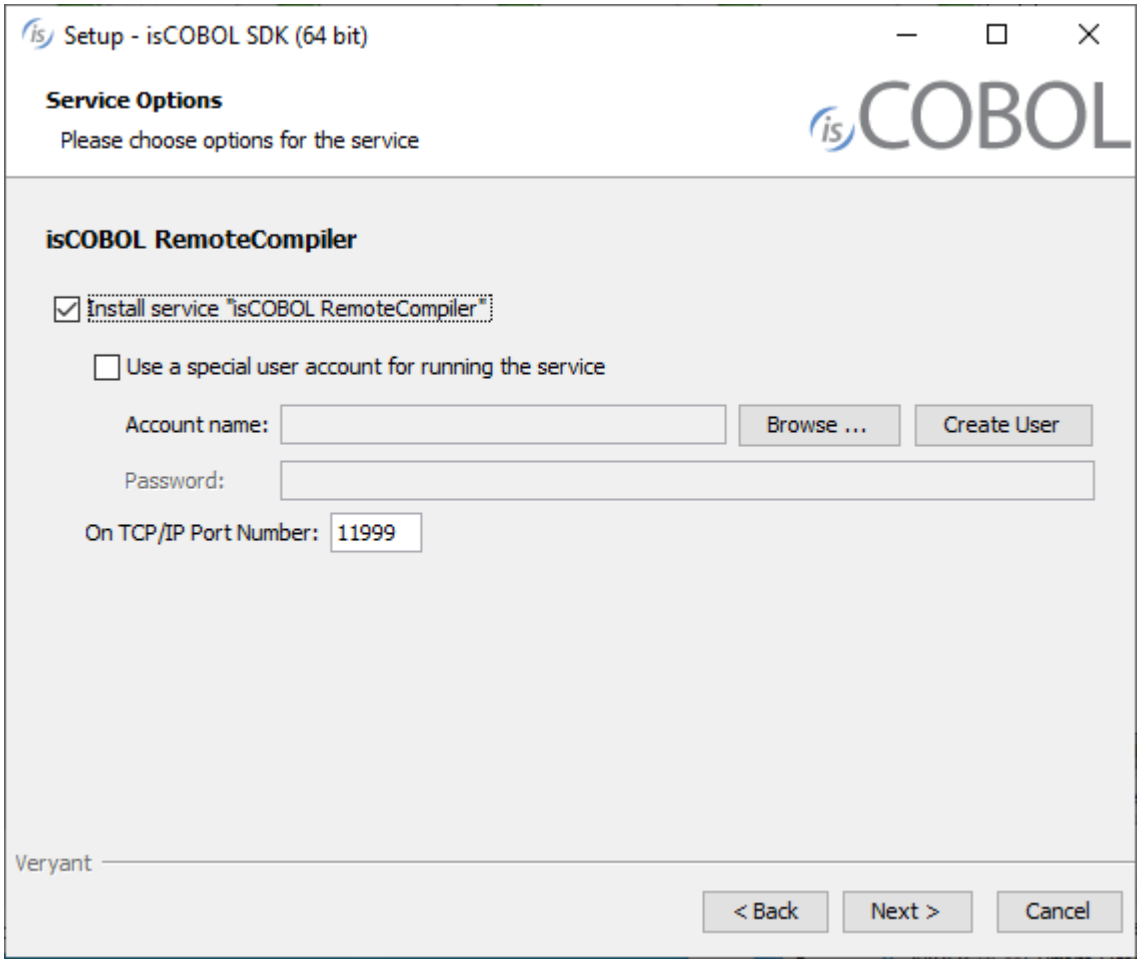


8. Select your JDK when prompted



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.
10. You will also be asked if you want to install the RemoteCompiler as a system service or not. If you don't install the service, you will have to start the RemoteCompiler in foreground mode from a command prompt as explained in Server configuration. See Windows service and Unix daemon for details about

the system service.



Quiet mode

The isCOBOL SDK setup supports the msi quiet mode. Settings can be driven with a response file.

A response file is a text file with name-value pairs that represent installer variables.

A response file is generated automatically after an installation is finished. The generated response file is found in the *.install4j* directory of the isCOBOL SDK and is named *response.varfile*.

When an installer is executed, it checks whether a file with the same name and the *.varfile* extension can be found in the same directory and loads that file as the response file. For example, if an installer is named *foo_setup.msi* on Windows, the response file next to it has to be named *foo_setup.varfile*.

For more information about msi setups and their command line options, see Microsoft Standard Installer Command-Line Options.

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://support.veryant.com".
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_2023_R1_*_Linux.32.i586.tar.gz
tar -xvf isCOBOL_2023_R1_*_Linux.32.i586.tar
```

   on Linux 64 bit:

```
gunzip isCOBOL_2023_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2023_R1_*_Linux.64.x86_64.tar
```

   on FreeBSD:

```
gunzip isCOBOL_2023_R1_*_FreeBSD.64.tar.gz
tar -xvf isCOBOL_2023_R1_*_FreeBSD.64.tar
```

   on Mac OSX:

```
gunzip isCOBOL_2023_R1_*_MacOSX.64.x86_64.tar.gz
tar -xvf isCOBOL_2023_R1_*_MacOSX.64.x86_64.tar
```

   on SunOS:

```
gunzip isCOBOL_2023_R1_*_SunOS.64.tar.gz
tar -xvf isCOBOL_2023_R1_*_SunOS.64.tar
```

7. Change to the "isCOBOL2023R1" folder and run "./setup", you will obtain the following output:

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

                          isCOBOL EVOLVE Installation
                          For isCOBOL Release 2023R1
                       Copyright (c) 2005 - 2023 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 LoadBalancer............................. (no)
    [9] isCOBOL Mobile................................... (no)

Install Path:
    [P] isCOBOL parent directory: UserHome/veryant

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 "4", then press Enter to select isCOBOL RemoteCompiler.
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://support.veryant.com".
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_2023_R1_*n*_noarch.tar.gz, where *n* is the build number.

Extract all contents of the archive:

```
gunzip isCOBOL_2023_R1_*_noarch.tar.gz
tar -xvf isCOBOL_2023_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.

The isCOBOL RemoteCompiler looks for the following configuration property for license keys:

```
iscobol.compiler.license.2023=<license_key>
```

The key 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. a custom configuration file passed on the command line
5. %ISCOBOL%\iscobol.properties

Unix/Linux

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

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.

Server configuration

The isCOBOL RemoteCompiler is activated by the following command on the server machine:

```
iscremotecc [-J-Discobol.remotecompiler.conf=configfile]
```

The RemoteCompiler configuration file is loaded by default from the user home directory. The default configuration file is: $USER_HOME/remoteCompiler.xml. Set the iscobol.remotecompiler.conf property to specify a different configuration file.

The configuration file has the following structure:

```
<!ELEMENT remoteCompiler (preProcessor+)>
<!ATTLIST remoteCompiler
    portNumber CDATA #IMPLIED
    outputFolder CDATA #IMPLIED
    cleanOutputFolderWhenExit (true | false) "true"
    deploymentFolder CDATA #IMPLIED>

    <!ELEMENT preProcessor (optionList,environment*)>
    <!ATTLIST preProcessor
        name CDATA #REQUIRED
        executable CDATA #REQUIRED
        outputFileExt CDATA #IMPLIED
        listFileExt CDATA #IMPLIED
        errorFileExt CDATA #IMPLIED>

        <!ELEMENT environment (variable*)>
        <!ATTLIST environment
            append (true | false) "true">

        <!ELEMENT variable EMPTY>
        <!ATTLIST variable
            name CDATA #REQUIRED
            value CDATA #REQUIRED>

        <!ELEMENT optionList (option+)>
        <!ELEMENT option (#PCDATA)>
        <!ATTLIST option
            if (listing|error) #IMPLIED
```

The main tag <remoteCompiler> describes the RemoteCompiler server. The following attributes are available:

- *portNumber* is the number of the port where the RemoteCompiler listens for client connections. If omitted, the port 11999 is used.
- *outputFolder* is the directory where translated source files and compiled classes will be stored. If omitted, the user TEMP folder is used.
- *cleanOutputFolderWhenExit* specifies if files should be deleted once they've been sent to the client. The default behavior is "true" in cases where this flag is omitted.
- *deploymentFolder* is a folder on the server where class files are stored after the compilation. Classes are sent to the client in any case, but, if this attribute is specified, a copy of these classes is kept on the server also. If the attribute is not specified, then classes are just sent to the client.

The <preProcessor> tag describes a preprocessor. Multiple <preProcessor> tags can appear as a child of the <remoteCompiler> tag. For each <preProcessor> the following attributes are available:

- *name* is a logical name that identifies the preprocessor. You can use any name here. This setting is mandatory.
- *executable* is the executable file of the preprocessor. This setting is mandatory.
- *outputFileExt* is the extension of the translated files. If omitted, the default value "cbl" is used.
- *listFileExt* is the extension of the list files if the preprocessor supports listing. If omitted, the default value "list" is used.
- *errorFileExt* is the extension of the error files if the preprocessor supports error files. If omitted, the default value "err" is used.

Each <preProcessor> tag can contain a list of environment variables and a list of command-line options that configure the preprocessor executable behavior.

Environment variables are listed in the <environment> tag, where the following attribute is available:

- *append*: if true the environment variables are appended to the existing environment, otherwise they replace it.

Each variable is identified by the <variable> tag with the two mandatory attributes *name* and *value*.

Options are listed in the <optionList> environment. Each option is identified by the <option> tag. The tag content may contain the keywords ${inputfile}, ${outputfile}, ${errorfile}, ${listingfile}. The content of the <option> tags is concatenated with the value of the <preprocessor> *executable* attribute. If the *if* attribute is specified and its value is "listing", the content of the tag is added to the command string only if the client request specifies the generation of the listing files. If its value is "error", the content of the tag is added to the command string only if the client request specifies the generation of the error files. The *if* condition is true when the corresponding isCOBOL option (*-lf* for listing and *-e* for errors) appears in the client compiler command line.

The following symbols are available for use in the <option> tag:

- ${inputfile} is automatically set to the name of the source file sent by the client.
- ${outputfile} is automatically set to the name of the translated file.
- ${listingfile} is automatically set to the name of the listing file.
- ${errorfile} is automatically set to the name of the error file.

The following snippet shows an example of usage of the <option> tag with the *if* attribute:.

```
<option if="list">-listopt ${listingfile}</option>
<option if="err">-erropt ${errorfile}</option>
```

The following sample configuration shows how to precompile with proCOBOL:

```
<remoteCompiler portNumber="12345" cleanOutputFolderWhenExit="true">
    <preProcessor name="procob"
        executable="/usr/local/bin/procob"
        outputFileExt="cob">
        <optionList>
            <option>-iname=${inputfile}</option>
            <option>-oname=${outputfile}</option>
        </optionList>
    </preProcessor>
</remoteCompiler>
```

The following sample configuration shows the minimal settings required for a pure COBOL remote compilation, without using precompilers.

```
 <remoteCompiler portNumber="12345" cleanOutputFolderWhenExit="true"
deploymentFolder="/opt/myCobolApp/programs"/>
```

User Authentication

If iscobol.as.authentication * is set to 2 in the server configuration, users will be prompted to provide login information at each compilation.

Client configuration

In order to perform a remote compilation, the following settings must be active in the client configuration:

```
iscobol.remotecompiler.host=servername
iscobol.remotecompiler.preprocnames=preprocessors
```

Where:

- *servername* is the name or the IP address of the machine where the RemoteCompiler is listening
- *preprocessors* is the list of preprocessors that will be executed. All the values of the name attribute of the <preProcessor> tag in the server configuration file are valid. Multiple values must be separated by comma. The following case insensitive special values are supported:

Value	Effect
ALL	all preprocessors defined in the Server configuration are executed.
NONE	none of the preprocessors defined in the Server configuration is executed. A simple COBOL compilation is performed. In this case the property iscobol.remotecompiler.compileonserver (boolean) is implicitly set to true.

With the above settings, each time you invoke the isCOBOL Compiler, it will compile remotely instead of locally. The objects that it receives from the server will be stored according to the -od option in the compiler command line on the client.

These additional settings are available client side:

- iscobol.remotecompiler.port

- iscobol.remotecompiler.compileonserver (boolean)
- iscobol.remotecompiler.createerrorfiles (boolean)
- iscobol.remotecompiler.createlistingfiles (boolean)
- iscobol.remotecompiler.preprocnames
- iscobol.remotecompiler.translateddir

The following sample configuration shows how to precompile using proCOBOL on the server and retrieve the translated source to be compiled on the local PC:

```
iscobol.remotecompiler.host=myserver
iscobol.remotecompiler.port=12345
iscobol.remotecompiler.preprocnames=procob
```

The following sample configuration shows how to compile programs remotely keeping the resulting classes on the server. Compiled classes are sent back to the client anyway:

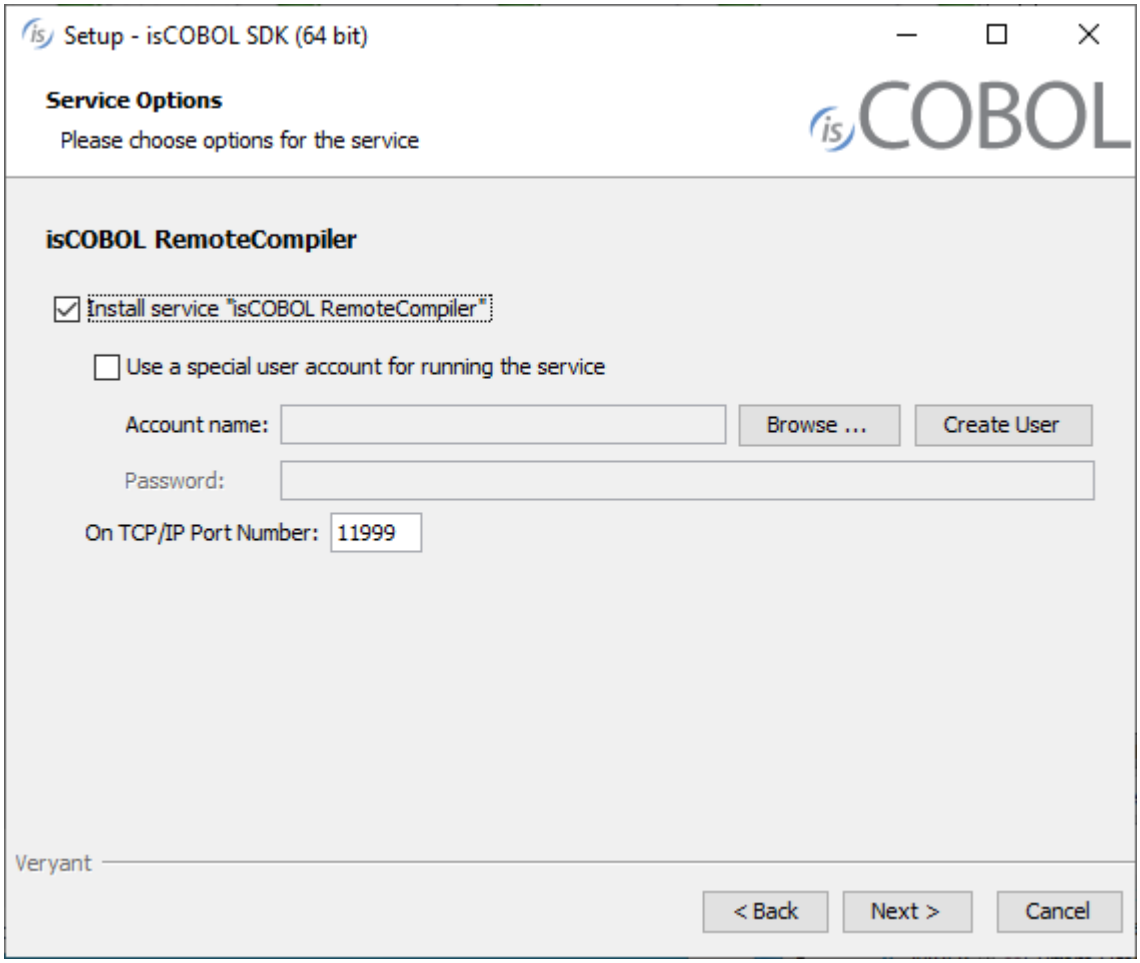
```
iscobol.remotecompiler.host=myserver
iscobol.remotecompiler.port=12345
iscobol.remotecompiler.preprocnames=NONE
iscobol.remotecompiler.compileonserver=1
```

Windows service and Unix daemon

Windows service

On Windows it's possible to install isCOBOL RemoteCompiler as a Windows Service.

The isCOBOL RemoteCompiler service can be installed during the setup process:



When isCOBOL has been installed, the service can be installed, removed and managed through the isremotecc.exe command line utility.

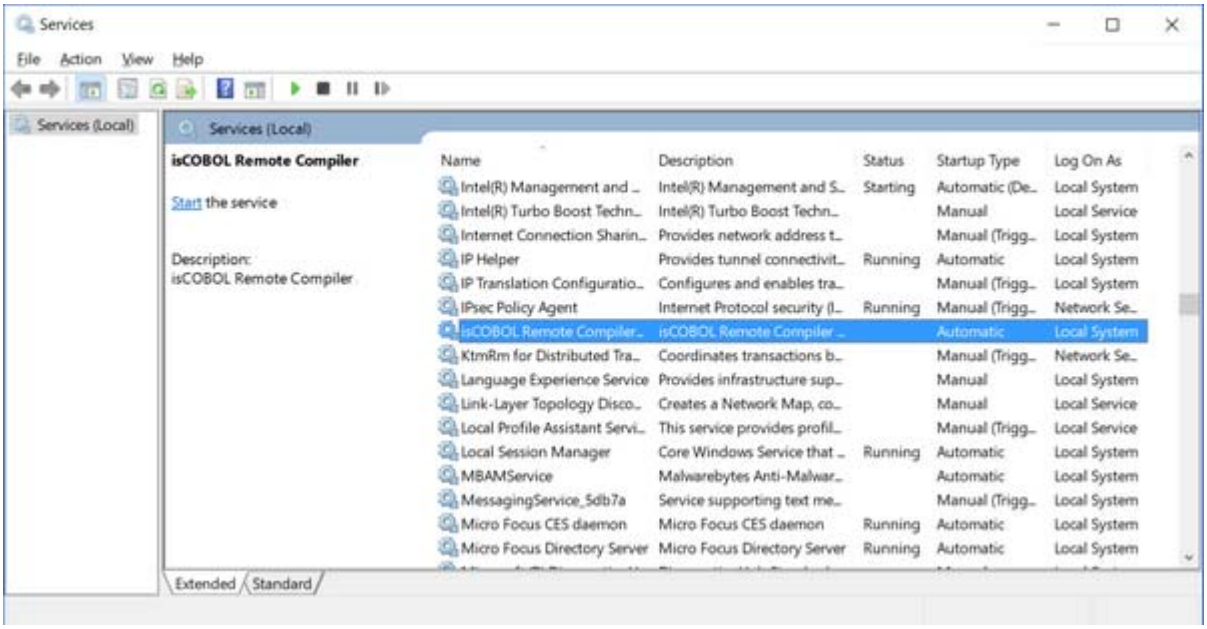
isremotecc.exe usage

The service maintenance is done through isremotecc.exe.

To install the service, use the command:

```
isremotecc -install
```

If the operation is successful, there will be a new entry in the Windows service manager.



The service is installed in auto mode, which means the service will automatically start along with the system.

To install the service in demand mode, use the command:

```
isremotecc -install-demand
```

In this mode, the service must be manually started by the user in the Windows service manager.

To retrieve the service status, use the command:

```
isremotecc -status
```

The exit code of this command is 0 when the service is running, 3 when it is not running and 1 when the state cannot be determined.

To start the service, use the command:

```
isremotecc -start
```

To stop the service, use the command:

```
isremotecc -stop
```

To uninstall the service, use the command:

```
isremotecc -uninstall
```

If the command is successful, the isCOBOL RemoteCompiler service will disappear from the Windows service manager.

In some situations, you might want to install a Windows service as a non-interactive service so that the service does not have any possibility to access the GUI subsystem. In order to do that, add the phrase non-interactive after the -install parameter. A custom service name can still be specified after the non-interactive parameter:

```
isremotecc -install non-interactive
```

It's also possible to specify a custom name for the service. This name should be added as last parameter of isserver.exe command line for all the options. For example, the following list of commands manges an isCOBOL RemoteCompiler service named "myservice":

```
isremotecc -install myservice
isremotecc -start myservice
isremotecc -status myservice
isremotecc -stop myservice
isremotecc -uninstall myservice
```

Output redirection

The isCOBOL RemoteCompiler service redirects all the console output (stderr and stdout) to two files named *isremotecc_err.log* and *isremotecc_out.log*. These files are located in the isCOBOL bin directory, which is the default directory of the service.

Service configuration

Java options must be put in the *isremotecc.vmoptions* file, located in the isCOBOL bin directory, which is the default directory of the service. In this file comments are prefixed by a hash and each option is on a separate line.

The following snippet shows how to configure memory limits, pass a custom configuration file and alter the Classpath for the isCOBOL RemoteCompiler service:

```
#memory settings
-Xmx256m
-Xms128m

#configuration
-Discobol.conf=/myapp/myconf

#classpath
-classpath/p .
-classpath/a C:\dev\myclasses.jar
```

The isCOBOL RemoteCompiler service inherits the Classpath from the system and adds all jar libraries in the isCOBOL lib directory to it. Using the *-classpath* option you can add additional items to the active Classpath. The value of *-classpath/p* is prepended to the active Classpath. The value of *-classpath/a* is appended to the active Classpath.

Note: On some Windows distributions it's necessary to reboot the system in order to make services aware of modifications to the system environment.

isCOBOL configuration properties to configure port number, hostname, rundebug, etcetera, can be set either in *isremotecc.vmoptions* with the syntax "-Dproperty=value" or in a file named *iscobol.properties* that will be loaded from:

1. The \etc directory
2. The user home directory

3. The Classpath

Unix daemon

On Unix systems, the isCOBOL RemoteCompiler can be installed as a daemon process and maintained using the isremotecc command.

isremotecc usage

The isremotecc command has the following options:

run	Run the isCOBOL RemoteCompiler service in foreground mode
start	Run the isCOBOL RemoteCompiler service in background mode
stop	Stop the isCOBOL RemoteCompiler service
restart	Restart the isCOBOL RemoteCompiler service
status	Show the status of the isCOBOL RemoteCompiler service

You need to be root in order to use this command.

Daemon configuration

The isremotecc command on Linux look for two files located in the bin directory:

- *default_java.conf* includes the paths to the isCOBOL and Java installations:

Variable	Description
ISCOBOL	isCOBOL installation directory
ISCOBOL_JDK_ROOT	JDK installation directory
ISCOBOL_JRE_ROOT	JRE installation directory

- *isremotecc.options* includes environment variables that can alter the command behavior:

Variable	Description
ISREMOTECC_HOME	Home directory of the command
ISREMOTECC_OPTS	Command line options for the command. Use a space to separate multiple options. When the options string contains spaces, delimit it with quotes, for example: ISREMOTECC_OPTS="-port 1234 -c my.properties"
ISREMOTECC_JAVA_EXE	Alternative JVM for the command
ISREMOTECC_JAVA_OPTS	Java options for the command. Use a space to separate multiple options. When the options string contains spaces, delimit it with quotes, for example: ISREMOTECC_JAVA_OPTS="-XX:+HeapDumpOnOutOfMemoryError -Xmx4G"

Variable	Description
ISREMOTECC_LOG_FILE	File where the command output is redirected
ISREMOTECC_PID_FILE	Semaphore file to inform if the command is running or not. This file contains the process ID.

These files are generated by the setup process.

In these files, comments are prefixed by a hash and each entry is on a separate line.

Environment variables in *isremotecc.options* can alternatively be set also in the external environment through the export command. The external environment has priority over the content of *isremotecc.options*.

Chapter 4

Utilities

The isCOBOL Evolve suite provides a number of utilities. The table below lists the available utilities telling if they should be used during development or in a runtime environment. It also tells if they can be launched in thin client environment via the -utility option of the isCOBOL Client.

Some of these utilities are affected by dedicated configuration properties. See Utilities Configuration for details.

Utility	Development	Runtime	Thin Client
AXC (ActiveX Compiler)	x		
COBFILEIO	x		x
CPGEN	x		
CPK (Color Picker)	x		x
GIFE (Index and Relative File Editor)		x	x
ISCONFIG	x		x
ISL (isCOBOL Launcher)		x	
ISMIGRATE (Index File Migration)		x	x
ISSORT (External Sort)		x	
ISUPDATER (Update Facility)		x	
JDBC2FD	x		x
JOE		x	
JUTIL		x	x
STREAM2WRK	x		
WSDL2WRK	x		
XML2WRK	x		x

AXC (ActiveX Compiler)

The ActiveX Compiler allows isCOBOL to use with ActiveX components via the JavaBean technology. It creates some bridge classes to be used along with the comfyj commercial software.

Contact Veryant for further information.

COBFILEIO

The COBFILEIO utility works together with the isCOBOL Compiler to read COBOL source code and generate Java classes that can be used to access COBOL files and records.

COBFILEIO reads External File Description (EFD) XML files that are produced by the isCOBOL Compiler when it has been executed with the -efd compiler option. An EFD is a data dictionary that contains the mapping to use when COBOL files and records are accessed externally. COBFILEIO reads two files, an EFD file and an FD file containing the standard COBOL file descriptor, and generates a Java class for the COBOL file and a Java class for the COBOL record.

The Java programmer does not need any knowledge of COBOL data types or their underlying storage format, and COBFILEIO automatically generates Javadocs for the file and record classes.

The resulting classes can be brought into any Java development environment and Java developers can take advantage of rapid development features such as Eclipse's "code assist" to pop-up documentation and provide single key or click code completion.

A data record is managed as an object with set and get accessor methods for each elementary field. The individual record fields are accessed using Java data types. The set methods automatically perform data validation and throw customizable exceptions.

COBFILEIO generates Object-Oriented COBOL source code for the classes. This source code can be customized and maintained in either COBOL or Java language.

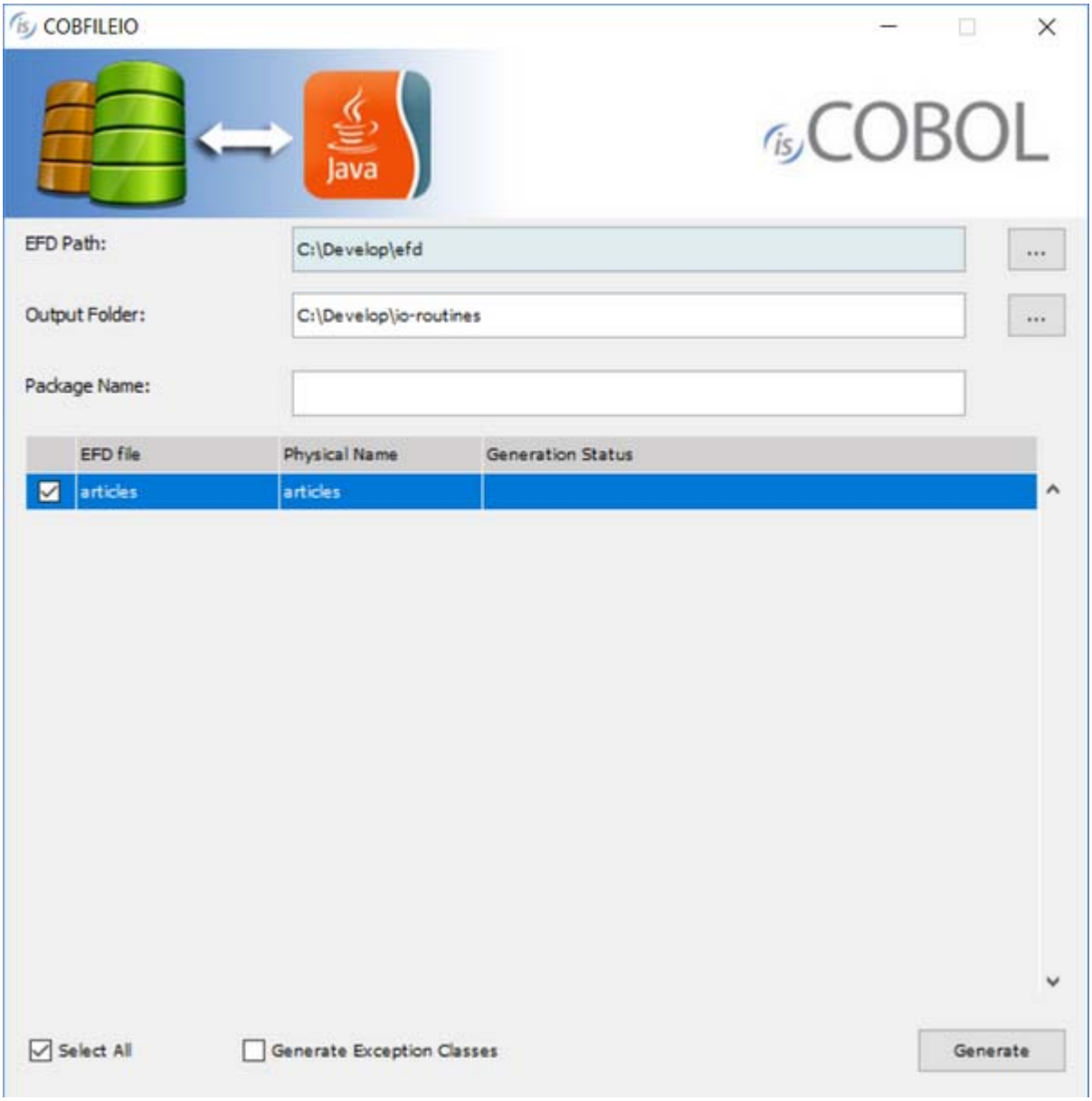
Usage 1:

```
cobfileio
```

or

```
iscrun -utility cobfileio
```

If the utility is launched without parameters, a graphical wizard procedure will start.



Usage 2:

```
cobfileio -help
```

or

```
iscrun -utility cobfileio -help
```

The *-help* option displays the usage on the system output.

Usage 3:

```
cobfileio fileName [-p=fileName] [-e]
```

or

```
iscrun -utility cobfileio fileName [-p=fileName] [-e]
```

Where:

- *fileName* is the external file name in all lowercase letters. For example, if in the file's SELECT statement there is ASSIGN TO DISK "/mydir/MYFILE", then the COBFILEIO command line would be "java COBFILEIO myfile".
- The -p option allows you to assign a custom physical file name, otherwise the same name as the EFD dictionary is used.
- The –e option causes COBFILEIO to generate the exception classes. This needs to be done only for the first file because the same exception classes are reused for every file.

Usage Steps

1. Create the EFD file by compiling the COBOL program with the -efd compile option. If desired, add the -efo=DirName compiler option to specify the directory where the EFD file will be output.
2. Execute the COBFILEIO utility with java COBFILEIO fileName -e. Include the -e option only the first time you run COBFILEIO.
3. Compile the exceptions classes. For example, javac *.java.
4. Compile the generated COBOL object classes. First compile the record class, FileNameRec.cbl. Then compile the file class, FileNameFile.cbl. If desired, add the -jj and -jc compiler options to generate Java source code.
5. If desired, use the javadoc utility that comes with the JDK to create Javadocs.

NOTE - By default, COBFILEIO attempts to read an EFD named fileName.xml from the current working directory. If your EFD file is in another directory then specify this directory as the value of the *cobfileio.efd_path* property. For example,

```
iscrun -c cobfileio.properties -utility cobfileio fileName
```

Where cobfileio.properties contains

```
iscobol.cobfileio.efd_path=../efd
```

Thin Client

COBFILEIO can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility cobfileio <arguments>
```

Server side paths must be provided in the arguments.

Configuration Properties

COBFILEIO has a number of properties that can be set in the configuration. Refer to the COBFILEIO section in Utilities Configuration for the list of available configuration properties.

API Reference in Javadoc Format

Use the following steps to create Javadocs for the file and record class generated by COBFILEIO:

1. Compile the COBOL source code for the record and file classes with the -jj and -jc command line options. For example,

```
iscc -jj -jc CustRec.cbl
iscc -jj -jc CustFile.cbl
```

2. Run the javadoc utility. For example,

```
javadoc -d htmlDir *.java
```

3. Double-click on the resulting index.html or open it with your favorite web browser

See the Javadoc Tool Home Page at https://www.oracle.com/java/technologies/javase/javadoc-tool.html for more information about the Javadoc Tool.

CPGEN

The CPGEN utility generates definition files for JavaBean events and properties.

An isCOBOL program must contain appropriate definition files to access JavaBean events and properties. CPGEN allows you to automatically create these definition files. A separate definition file will be created for each class to which the JavaBean extends, directly or indirectly.

Usage.

```
cpgen [-p package] cls1 [ cls2 ... clsN] [-d outputDir]
```

Example:

To use the JCalendar bean (com.toedter.calendar.JCalendar in jcalendar.jar), you must create the definition files with one of the following commands:

```
cpgen -p com.toedter.calendar JCalendar
```

or

```
cpgen com.toedter.calendar.JCalendar
```

Since the sructure of JCalendar is

```
java.lang.Object
  byjava.awt.Component
    byjava.awt.Container
      byjavax.swing.JComponent
        byjavax.swing.JPanel
          bycom.toedter.calendar.JCalendar
```

the following definition files will be created:

object.def
component.def
container.def
jcomponent.def
jpanel.def
jcalendar.def

The resulting definition files will contain event definitions

```
*>   KEY event definitions, Class: java.awt.event.KeyEvent.
78 COMPONENT-KEYPRESSED VALUE 1024581858.
78 COMPONENT-KEYRELEASED VALUE 274742396.
78 COMPONENT-KEYTYPED VALUE 1303301483.
```

and the list of the available properties

```
*>   Control Properties.
*> NAME: BACKGROUND , TYPE: OBJECT REFERENCE (java.awt.Color) W
*> NAME: CALENDAR , TYPE: OBJECT REFERENCE (java.util.Calendar) R/W
*> NAME: DATE , TYPE: OBJECT REFERENCE (java.util.Date) R/W
*> NAME: DAYCHOOSER , TYPE: OBJECT REFERENCE (com.toedter.calendar.JDayChooser) R
*> NAME: DECORATIONBACKGROUNDCOLOR , TYPE: OBJECT REFERENCE (java.awt.Color) R/W
*> NAME: DECORATIONBACKGROUNDVISIBLE , TYPE: NUMERIC INTEGER [VALUES 0/1] (boolean) R/W
*> NAME: DECORATIONBORDERSVISIBLE , TYPE: NUMERIC INTEGER [VALUES 0/1] (boolean) R/W
*> NAME: ENABLED , TYPE: NUMERIC INTEGER [VALUES 0/1] (boolean) R/W
*> NAME: FONT , TYPE: OBJECT REFERENCE (java.awt.Font) W
*> NAME: FOREGROUND , TYPE: OBJECT REFERENCE (java.awt.Color) W
*> NAME: LOCALE , TYPE: OBJECT REFERENCE (java.util.Locale) R/W
*> NAME: MAXDAYCHARACTERS , TYPE: NUMERIC INTEGER (int) R/W
*> NAME: MAXSELECTABLEDATE , TYPE: OBJECT REFERENCE (java.util.Date) R/W
*> NAME: MINSELECTABLEDATE , TYPE: OBJECT REFERENCE (java.util.Date) R/W
*> NAME: MONTHCHOOSER , TYPE: OBJECT REFERENCE (com.toedter.calendar.JMonthChooser) R
*> NAME: SUNDAYFOREGROUND , TYPE: OBJECT REFERENCE (java.awt.Color) R/W
*> NAME: WEEKOFYEARVISIBLE , TYPE: NUMERIC INTEGER [VALUES 0/1] (boolean) R/W
*> NAME: WEEKDAYFOREGROUND , TYPE: OBJECT REFERENCE (java.awt.Color) R/W
*> NAME: YEARCHOOSER , TYPE: OBJECT REFERENCE (com.toedter.calendar.JYearChooser) R
```

Thin Client

CPGEN can't be launched directly by the isCOBOL Client.

CPK (Color Picker)

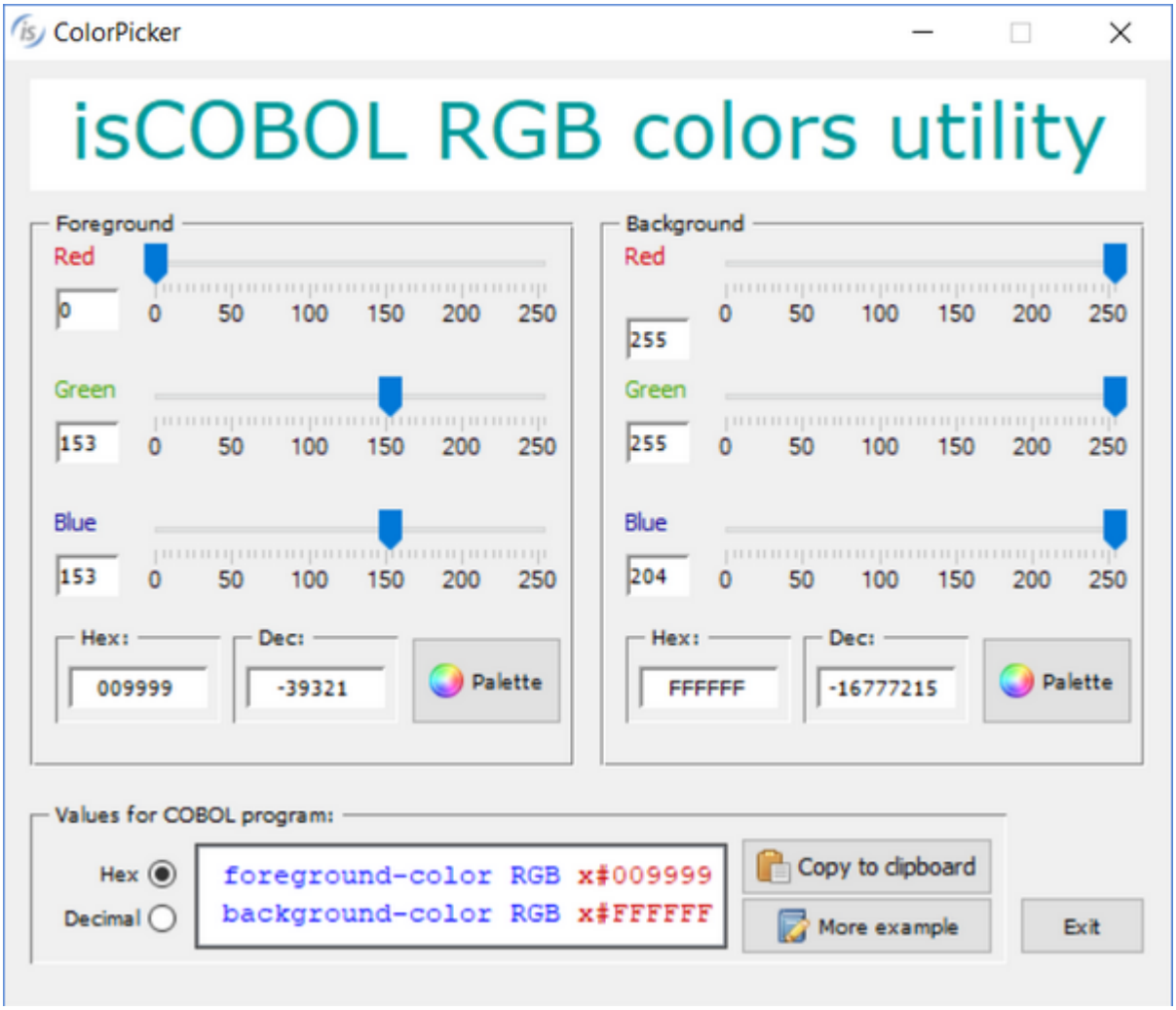
The Color Picker (CPK) utility allows you to easily calculate color values to be used by COBOL programs.

Usage:

```
cpk
```

or

```
iscrun -utility cpk
```



Drag the sliders or type numbers between 0 and 255 in the fields until you see the desired colors in the title box at the top of the dialog. The chosen color values are shown below, along with syntax snippets that you can copy to clipboard.

Thin Client

CPK can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility cpk
```

GIFE (Index and Relative File Editor)

The Graphical Indexed File Editor (GIFE) utility allows you to read and modify the content of indexed and relative files.

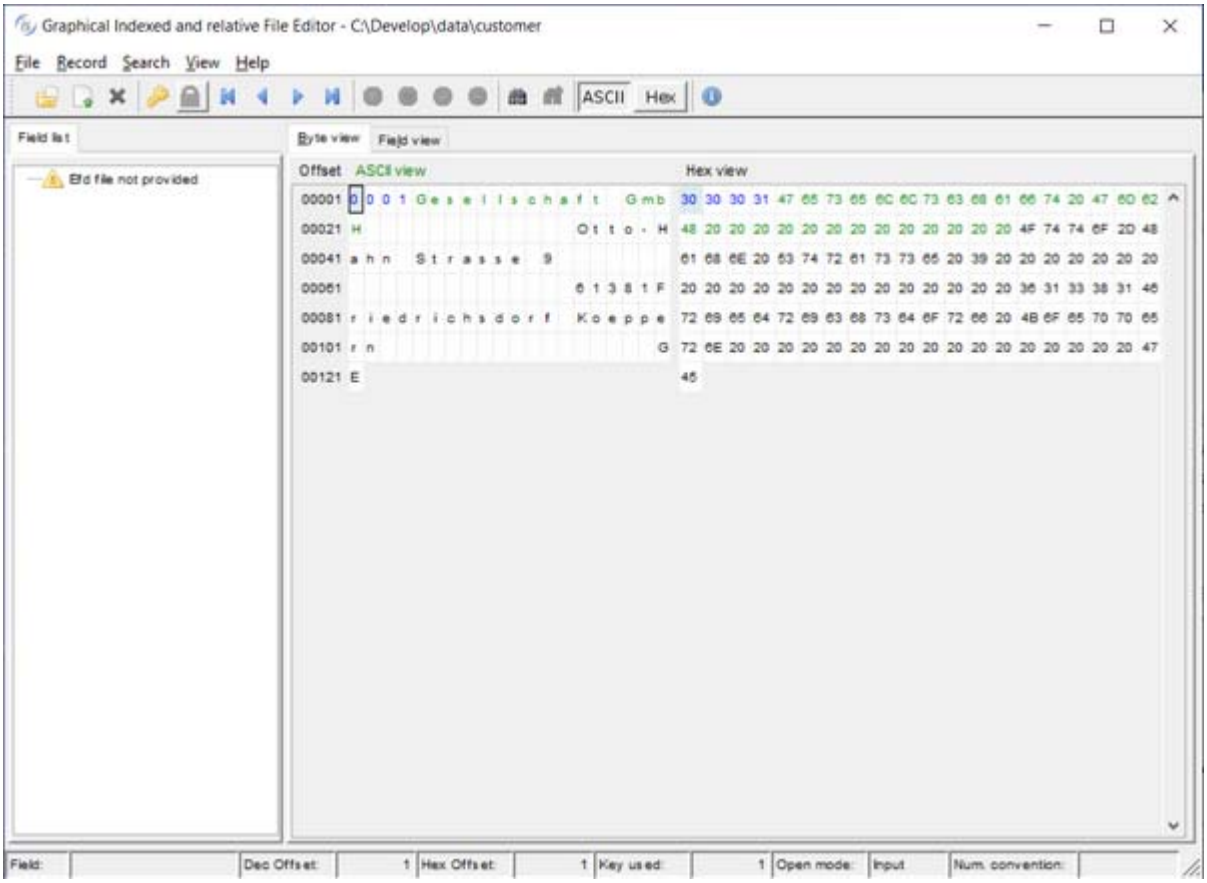
Indexed files where the record size is larger than 64KB are not supported by GIFE.

Usage 1:

```
gife
```

or

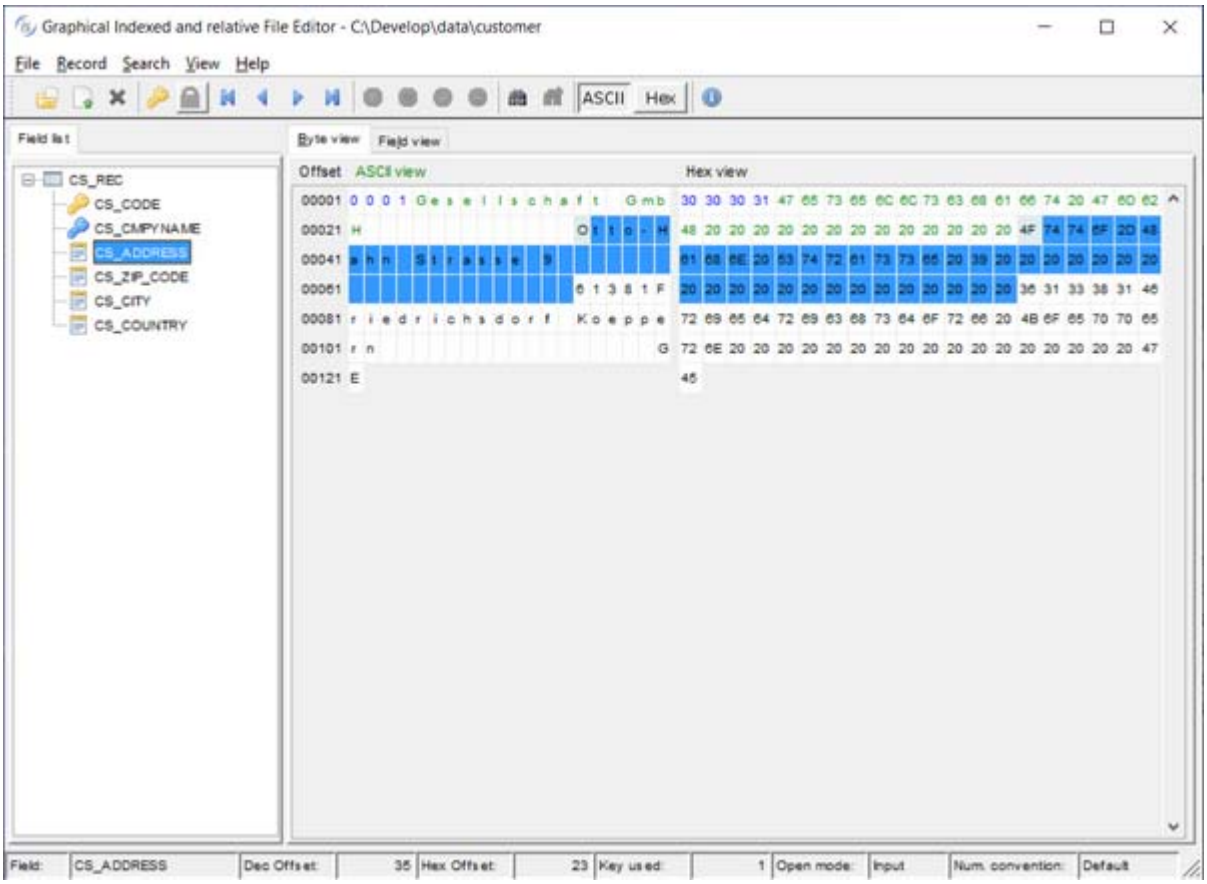
```
iscrun -utility gife
```

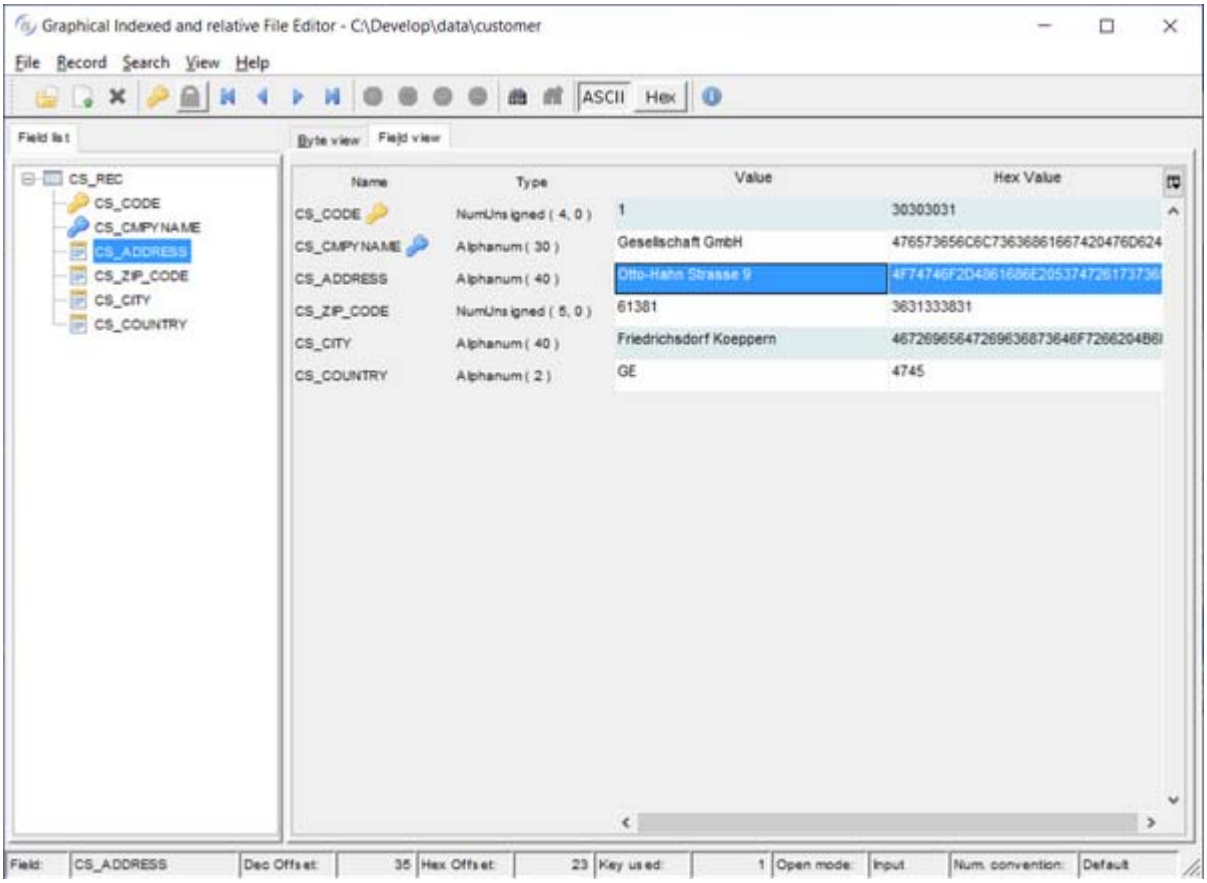


If the utility is launched without parameters, an empty dialog is shown.

To open a file click on the *File* menu and select *Open*. You will be prompted for the file name, file type and open mode (input / I-O). If iscobol.file.prefix is set in the configuration, the first path of the file prefix is proposed. If iscobol.file.index is set in the configuration, its value is proposed as file type. If you plan to open a relative file rather than an indexed file, set the file type field to "relative".

You can optionally specify an External File Description (EFD) XML file. This kind of file is produced by the isCOBOL Compiler when it has been executed with the -efd compiler option. An EFD is a data dictionary that contains the mapping to use when COBOL files and records are accessed externally. When provided with an EFD, GIFE shows the list of fields and allows you to work on each single field through two different views.





The program shows the first record as soon as the file is open. ASCII view of the record content is shown on the left; this view is useful to handle USAGE DISPLAY items. Hex view is shown on the right; this view is useful to handle USAGE COMP and other kind of items that can't be correctly represented in ASCII.

Primary key digits are shown in blue. Alternate keys digits are shown in green. The rest of the record is shown in black.

The *Record* menu contains features that allow you to navigate through records and update the record content.

From the *Search* menu you can perform a search for a specific word in the current record.

The *View* menu allows you to switch between ASCII view and HEX view.

The *Lock* button on the toolbar is used to lock and unlock the current record.

Usage 2

```
gife filename [ EFDfile ]
```

or

```
iscrun [-c gife.properties ] -utility gife filename [ EFDfile ]
```

Refer to the GIFE section in Utilities Configuration for the list of configuration properties that can be included in *gife.properties* to configure GIFE.

If you pass the name of a file as parameter on the command-line, GIFE opens the file automatically by using the handler set by the iscobol.file.index configuration property.

Thin Client

GIFE can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility gife <arguments>
```

Usage 1: server side paths must be provided in the arguments.

Usage 2: GIFE looks for files on the server machine. Browse features are disabled, you need to type the files path by hand.

ISCONFIG

The ISCONFIG utility converts an ACUCOBOL-GT® configuration file to an isCOBOL configuration file.

Usage:

```
isconfig fileIn [fileOut]
```

Where:

- *fileIn* is the name of the ACUCOBOL-GT configuration file to be translated.
- *fileOut* is the name of the resulting isCOBOL configuration file. If omitted, "iscobol.properties" is assumed.

Thin Client and Code Prefix:

ISCONFIG can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility isconfig <arguments>
```

Server side paths must be provided in the arguments.

ISL (isCOBOL Launcher)

The ISL utility helps setting up startup commands for your COBOL application.

Usage:

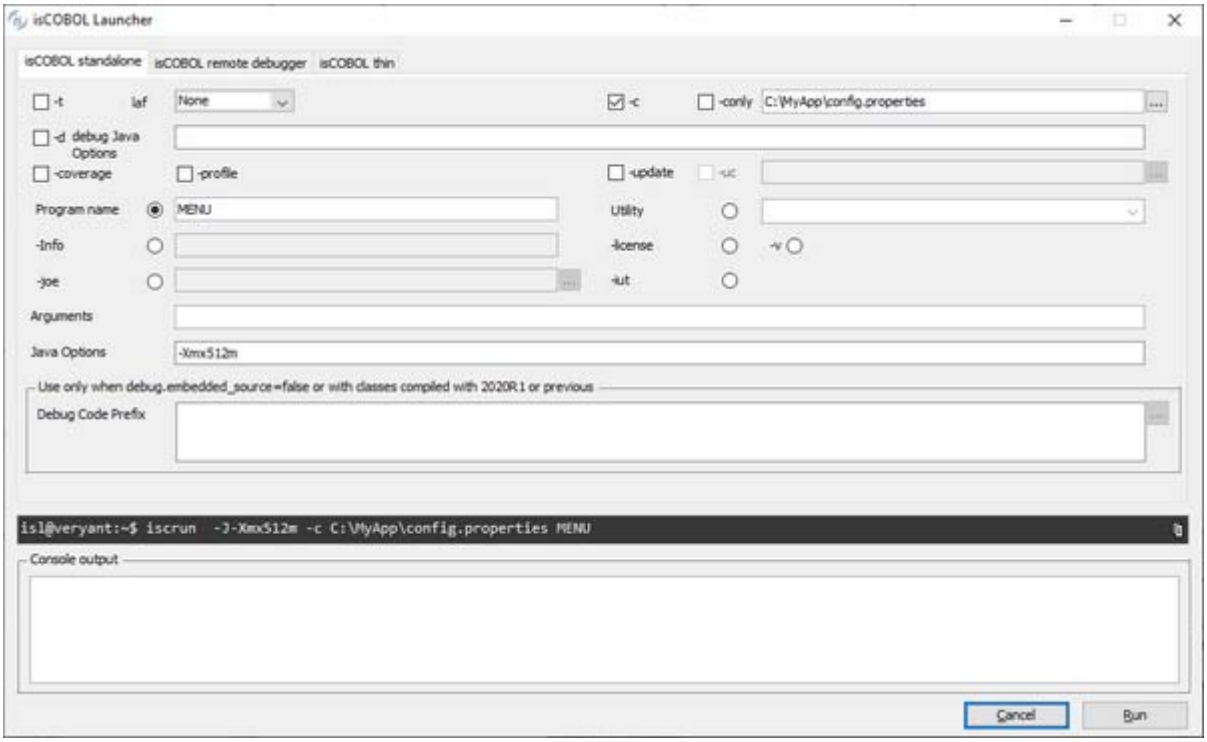
```
isl
```

or

```
iscrun -utility isl
```

Configuring isCOBOL standalone

In this screen you can set up the command to run a COBOL program in stand alone mode, with or without debug.



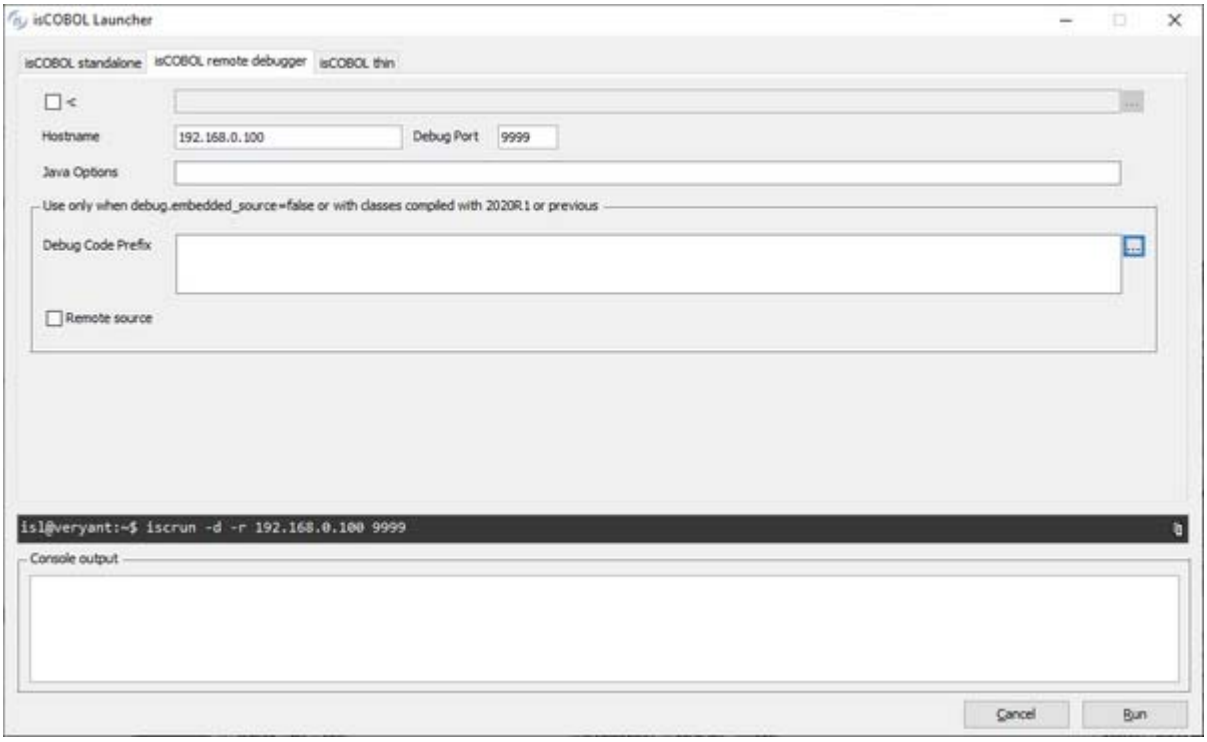
Available options	Description
-t	Run in terminal mode on Linux/Unix terminals
laf	Select the Look and Feel
-c	Use an additional runtime configuration file
-conly	Use only one runtime configuration file
-d	Run in debug mode
Debug Java Options	Options to be passed to the JVM that runs the Debugger
-coverage	Generate code coverage report of the runtime session
-profile	Profile runtime execution
-update	Look for updates before starting the runtime
-uc	Specifies a configuration file for the isUpdater utility, that is invoked by the -update option

Available options	Description
Program name	Name of the COBOL program to run
Utility	Run an utility instead of a COBOL program. Choose the desired utility from the list
-info	Instead of running the program, print information about the program class
-license	Instead of running the program, print information about the license
-v	Instead of running the program, print the runtime version
-joe	Run a joe script instead of a COBOL program
-iut	Run a unit test instead of a COBOL program
Arguments	Arguments to the COBOL program
Java options	Options to be passed to the Java Runtime
Debug Code Prefix	Path list to locate the source files when running in debug mode. This is useful only for debugging programs that were compiled with isCOBOL 2020 R1 or previous

While you fill the fields and check the options, the resulting command is shown in the dark entry-field on the bottom of the screen; you can copy it to the clipboard and paste it in a batch file using a text editor. Click the *Run* button to start the COBOL program.

Configuring isCOBOL remote debugger

In this screen you can set up the command to remotely debug a COBOL program, assuming that you know the IP address and port of a Debugger listener.

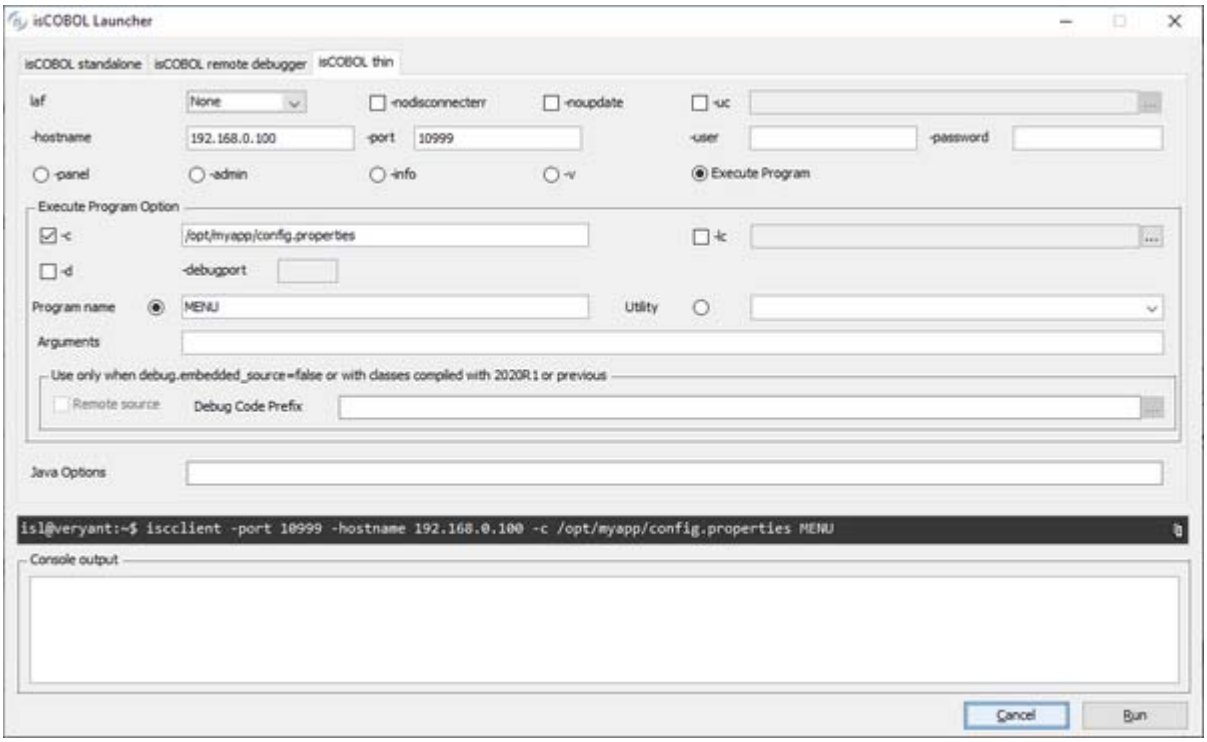


Available options	Description
-c	Use an additional runtime configuration file
Hostname	IP address or hostname that runs the program to remote debug
Debug Port	Port used by the remote host for debugging communication
Java Options	Options to be passed to the Java Runtime
Debug Code Prefix	Path list to locate the source files when running in debug mode
Remote source	The source files are located on the remote host

While you fill the fields and check the options, the resulting command is shown in the dark entry-field on the bottom of the screen; you can copy it to the clipboard and paste it in a batch file using a text editor. Click the *Run* button to start the COBOL program.

Configuring isCOBOL thin

In this screen you can set up the command to run or debug a COBOL program in thin client mode, assuming that you know the IP address and port of a listening Application Server. You can also run server administration utilities from here.



Available options	Description
laf	Select the Look and Feel
-nodisconnecterr	Avoid message box notification on connection lost
-noupdate	Don't check for updates before starting the isCOBOL Client
-hostname	IP address or hostname where the isCOBOL Application Server is running
-port	Port used by the isCOBOL Application Server running on the host
-user	Specifies the user for the connection to the isCOBOL Server
-password	Specifies the password for the connection to the isCOBOL Server
-panel	Run the isCOBOL Server's Administration Panel.
-admin	Run the isCOBOL Server's Administration Panel. The panel starts with the "Users View" tab active
-info	Print the list of connected clients on the console
-v	Print the isCOBOL Client version on the console
Execute program	Run a program in thin-client mode. It enables Execute program options
Java Options	Options to be passed to the Java Runtime

Execute program options

Available options	Description
-c	Use an additional runtime configuration file, server side
-lc	Use an additional runtime configuration file, client side
-d	Run in debug mode
-debugport	Port used by the remote host for debugging communication
Program Name	Name of the COBOL program to run
Utility	Run an utility instead of a COBOL program. Choose the desired utility from the list
Arguments	Arguments to the COBOL program
Remote source	The source files are located on the remote host. This is useful only for debugging programs that were compiled with isCOBOL 2020 R1 or previous
Debug Code Prefix	Path list to locate the source files when running in debug mode. This is useful only for debugging programs that were compiled with isCOBOL 2020 R1 or previous

While you fill the fields and check the options, the resulting command is shown in the dark entry-field on the bottom of the screen; you can copy it to the clipboard and paste it in a batch file using a text editor. Click the *Run* button to start the COBOL program.

Presets

ISL inherits most of the settings from the environment in which it's launched. For settings that cannot be inherited, some configuration properties are provided. Refer to the ISL section in Utilities Configuration for the list and description of these properties.

In order to run ISL with a configuration file (e.g. *isl.properties*), use the command:

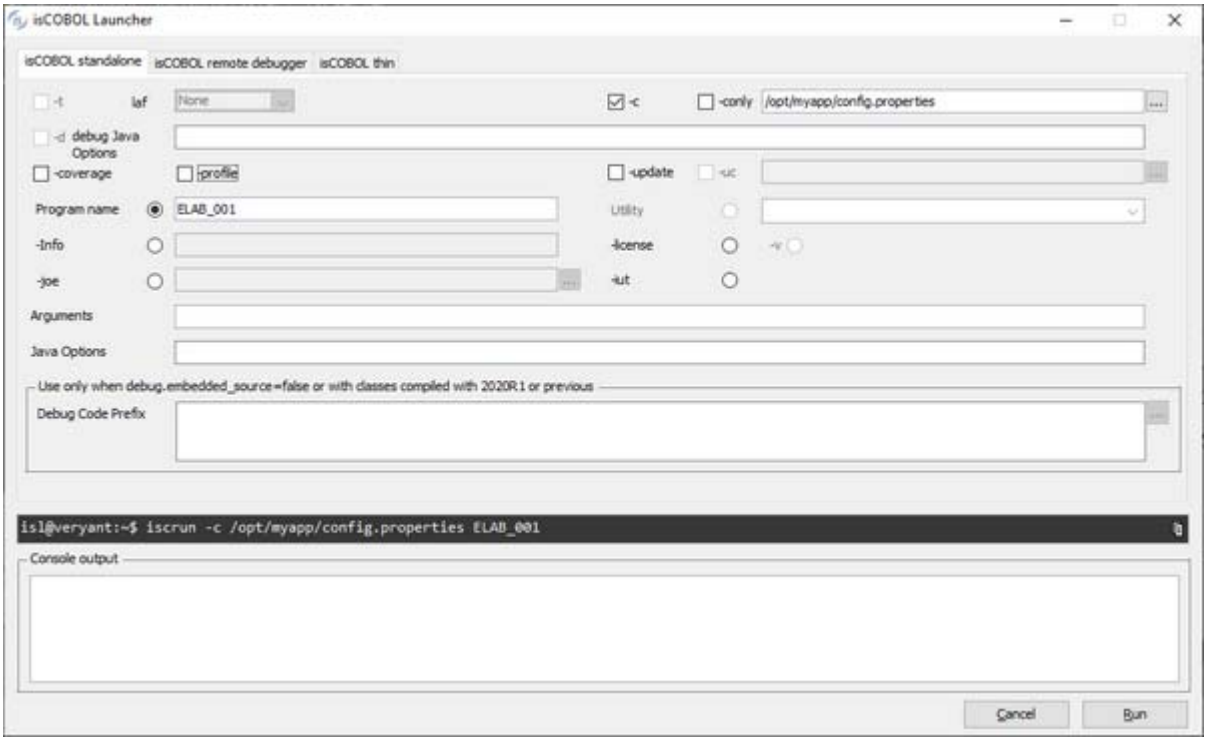
```
iscrun -c isl.properties -utility isl
```

Thin Client

ISL can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility isl <arguments>
```

When launched in thin client environment, the utility allows you to configure only the launch of a backend elaboration that will run server side. This means that only the 'isCOBOL standalone' tab is active and within that tab all the options related to the UI are disabled.



ISMIGRATE (Index File Migration)

The ISMIGRATE utility converts ISAM files from one format to another.

In order to perform the conversion, ISMIGRATE calls low level file handler functions. The isCOBOL Framework must be able to interact with the involved file handlers, so native libraries, licenses and specific configuration of the file handler must be available to run ISMIGRATE.

Before using ISMIGRATE it's good practice to verify the integrity of the files you're going to convert. Corrupted files will not migrate correctly.

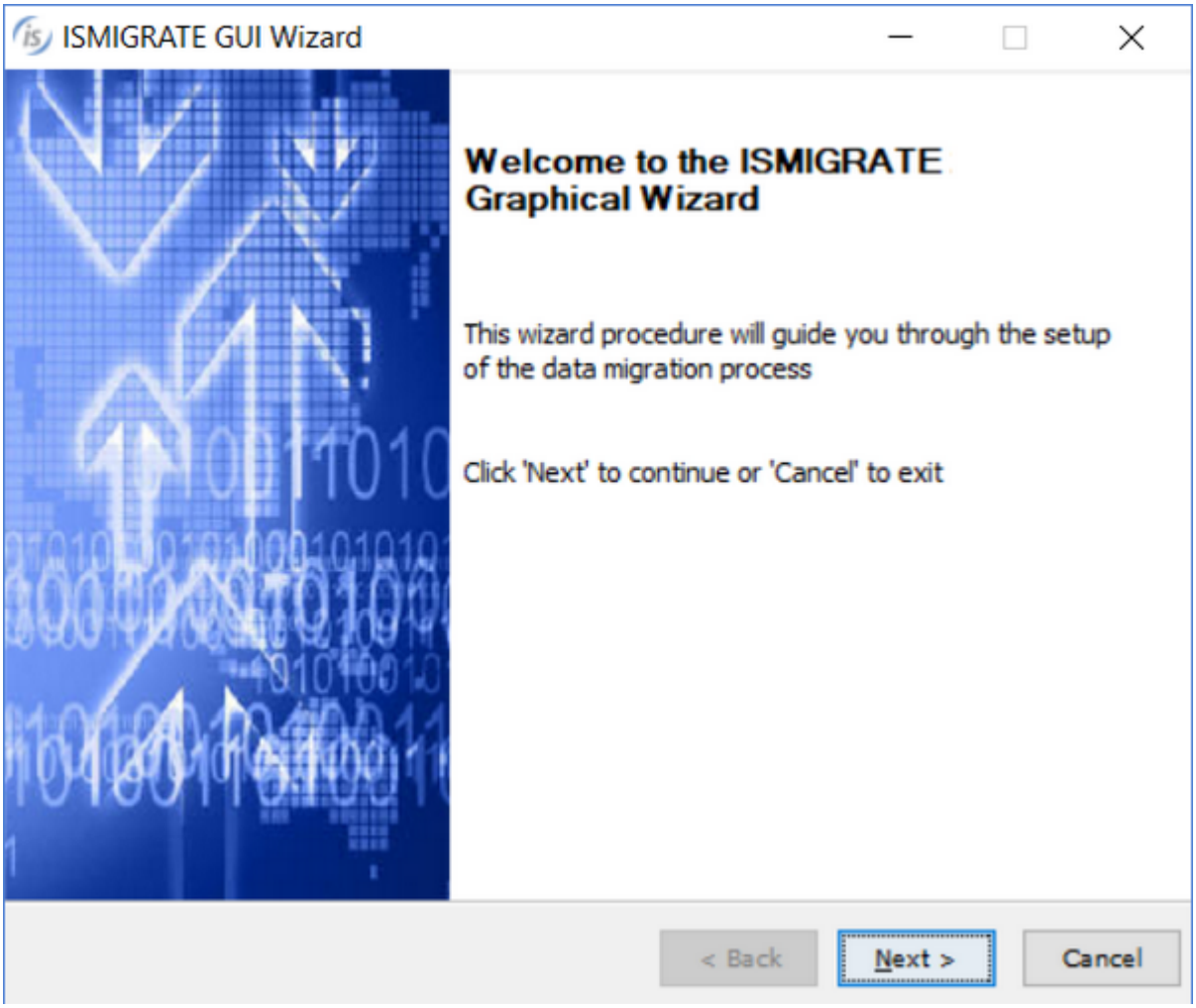
Usage 1:

```
ismigrate
```

or

```
iscrun -utility ismigrate
```

If the utility is launched without parameters, a graphical wizard procedure will start.



The wizard procedure allows you to set up the migration process through the following steps:

1. Choose input and output file systems: you can choose from the two lists that are loaded with the known file handlers. If you need to use a different file handler, just type its name into the field. The following file handlers are available by default:

File Handler	Description
c-tree RTG - Java Interface (ctreej)	Current c-tree file handler interface. The c-tree client library is required in the Java library path. When used as input file handler, it must be configured externally and Ismigrate will browse for c-tree files on the local PC. When used as output file handler, it can be configured in the Ismigrate panels and you're allowed to write c-tree files also on a remote machine. You're also allowed to sqlize files to have them registered as tables in the c-treeSQL database as long as you have iss dictionaries for these files.

File Handler	Description
CASEMaker DBMaker (dci)	DBMaker interface. The dci library is required in the Java library path. When used as input file handler, the DBMaker JDBC driver (dmjdbc30.jar) is required in the Classpath as Ismigrate uses it to get the list of available tables.
isCOBOL DatabaseBridge (easydb)	DatabaseBridge. The JDBC driver of the desired database is required in the Classpath. EDBI routines for the files that you're going to convert must be available either in the Classpath or in the iscobol.code_prefix.
isCOBOL JISAM (jisam)	Jisam.
ACUCOBOL Vision (com.iscobol.io.ScanVision)	Internal class that is able to read Vision files sequentially without the need of external libraries or connectors. This class is not able to read encrypted Vision files. **Note -** This class is deprecated, you should consider using *ACUCOBOL Vision - Java Interface (visionj)* instead.
ACUCOBOL Vision - Java Interface (visionj)	Vision interface. It provides read, write and lock capabilities on ACUCOBOL Vision files. It supports Vision 3, Vision 4, Vision 5 and Vision 6 files.
Micro Focus (com.iscobol.io.ScanMF)	Internal class that is able to read Micro Focus indexed files sequentially without the need of external libraries or connectors. This should be the first choice when you have to convert Micro Focus files. This class is not able to read compressed files. If this class fails to read your Micro Focus files, then you should consider trying with "isCOBOL Micro Focus File Connector (mfc)".
RM/COBOL (com.iscobol.io.ScanRMKF)	Internal class that is able to read RM/COBOL indexed files sequentially without the need of external libraries or connectors.
isCOBOL c-tree File Connector (fscsc)	File connector that allows you to access c-tree files using a separate process. See The c-tree File Connector for more details about the fscsc executable. There is no particular advantage in using this file handler for data migration. Consider using "c-tree RTG - Java Interface (ctreej)" instead.
isCOBOL Micro Focus File Connector (mfc)	File connector that allows you to access Micro Focus indexed files using a separate process. See The Micro Focus File Connector for more details about the mfc executable.
isCOBOL Vision File Connector (vfc)	File connector that allows you to access Vision files using a separate process. See The Vision File Connector for more details about the vfc executable.
Btrieve (btrieve)	Pervasive Btrieve file handler. The wbtrv32 library is required in the Java library path.

2. Choose input files: ISMIGRATE loads all files that can be open by the input file system from the specified input directory and allows you to check the one you wish to migrate. When migrating a database table, ISMIGRATE connects to the database through JDBC and lists available tables.
3. Choose output directory: unless you're migrating to a database, ISMIGRATE allows you to choose the directory in which resulting files will be stored.
4. Enable or disable additional options: this is the last step before the migration starts. Here you can enable the trace of the Ismigrate activity and the check of consistency between the original records and the new records. It's also possible to specify a hook program to be called for processing records before they're written to the output file.

5. Migration process: in the last panel you can see the data migration status. In the top area you can monitor the progress of the current file migration and have an estimated time of arrival (ETA) based on the writing speed; note that the progress is calculated by comparing the number of transferred records against the total number of records, it doesn't include the time spent for closing the files, so it might not be accurate if you have enabled bulk addition. In the bottom area you can review the list of completed file migrations; for each file migration Ismigrate provides the count of processed records and the status. If you check 'save session' before exiting, ISMIGRATE saves all the information gathered during the wizard procedure into a file with imgs extension that can be used with the -silent option (see Usage 3, below).

With this usage ISMIGRATE generates two log files in the isCOBOL bin directory: *ismigrate_out.log* and *ismigrate_err.log*. These files contains the output that ISMIGRATE prints on sysout and syserr.

Usage 2:

```
iscmigrate Input Output
```

or

```
iscrun [-c ismigrate.properties ] -utility ismigrate Input Output
```

If the utility is launched with parameters, it works in console mode without showing any window and displaying the output on the system output. Necessary information is read from the configuration (see ISMIGRATE Properties below). See ISMIGRATE Parameters below for details on *Input* and *Output*.

There are two different approaches that you can adopt when running ISMIGRATE on the command line:

- migrate one file at a time
  - The property iscobol.ismigrate_no_directories (boolean) must be set to true.
  - *Input* and *Output* specify the exact name of the input file to be read and the exact name of the output file to be written. Relative paths are resolved according to the working directory. Ensure to pass the correct COBOL file name, that sometimes doesn't match with the disk file name (e.g. don't specify the "dat" extension in the names of JIsam and c-tree files).
- migrate multiple files at a time
  - *Input* specify the name of a directory optionally followed by a pattern, e.g. "data*.dat". *Output* specifies the output directory.
  - ISMIGRATE performs a directory listing of *Input* and, for each file found, it creates a file with the same name in *Output*. ISMIGRATE uses the disk file name, that sometimes doesn't match with the COBOL file name required by I/O APIs. When the input files are JIsam or c-tree, you should instruct ISMIGRATE to discard the "dat" extension from their name; this is achieved through the iscobol.ismigrate_remove_extension configuration property.

During the file migration, Ismigrate prints the progress on the system output; note that the progress is calculated by comparing the number of transferred records against the total number of records, it doesn't include the time spent for closing the files, so it might not be accurate if you have enabled bulk addition.

Usage 3:

```
iscmigrate -silent:imgs_file
```

or

```
iscrun -utility ismigrate -silent:imgs_file
```

The -silent option makes ISMIGRATE run in background mode retrieving necessary parameter from a imgs file. See Usage 1 above to see how to generate imgs files.

Usage 4:

```
call "ISMIGRATE" USING Input
                       Output
                      [IsmigrateResult]
               GIVING IsmigrateStatus
```

The ISMIGRATE utility can also be called from within a COBOL program. In this case, it works in console mode without showing any window and displaying the output on the system output. Necessary information is read from the configuration (see ISMIGRATE Properties below). See ISMIGRATE Parameters below for details on *Input* and *Output*.

IsmigrateStatus is a signed numeric data item that receives the ISMIGRATE Exit Status.

Note - the ISMIGRATE utility is not thread safe. Calling multiple instances of the utility simultaneously in the same runtime session may lead to unexpected errors.

ISMIGRATE Parameters

- *Input* is the name of the file(s) to be converted, or the name of directory containing files to be converted. When passed on the command line, if Wild Card characters (* and ?) are used, the parameter must be within quotation marks (“).
- *Output* is the name of the directory that will receive the converted files. If this directory doesn't exist, it will be created.
- *IsmigrateResult* is an optional parameter that receives detailed information about the last file migrated. It must be defined as follows:

```
      01 more-info.
          03 read-count  long.
          03 write-count long.
          03 skip-count  long.
          03 check-count long.
          03 error-buffer.
             05 err      pic x(128).
             05 err-ext  pic x(256).
```

read-count is the number of records read,
write-count is the number of records written,
skip-count is the number of records skipped,
check-count is the number of records verified,
error-buffer is the error description, or spaces if everything is OK.

ISMIGRATE Properties

ISMIGRATE has a number of properties that can be set in the configuration. Refer to the ISMIGRATE section in Utilities Configuration for the list of available configuration properties.

ISMIGRATE Exit Status

The ISMIGRATE utility terminates with one of the following exit status

Status	Meaning
0	Operation Successful
-1	Bad Parameters
-2	Bad Environment
-3	I/O Error

Thin Client

ISMIGRATE can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility ismigrate <arguments>
```

Usage 1: the utility works on server files. Browse features are disabled, you need to write folder paths by hand.

Usage 2 and 3: server side paths must be provided in the arguments.

ISSORT (External Sort)

The ISSORT utility enables you to sort or merge indexed, relative, binary sequential, and line sequential files.

The utility internally uses the SORT verb, so it's affected by the configuration settings whose name starts with "iscobol.sort" (e.g. iscobol.sort.memsize). The utility uses the file handler specified in the configuration to sort a specific kind of file. For example, when sorting indexed files, the utility uses the file handler specified by the iscobol.file.index property. The activity of this utility is traced in the isCOBOL log if iscobol.tracelevel includes the value 8 (trace file activity).

Usage

ISSORT accepts a quite complex set of instructions in order to fulfill the user requirements.

The instructions may appear directly on the command line,

```
issort <instructions>
```

or they may be included in a separate text file

```
issort take <filename>
```

Passing instructions directly on the command line is appropriate if you want to execute a simple sort with few parameters. When you choose this method, you must ensure that the operating system conventions and limits are fulfilled.

Passing instructions through a text file is appropriate if you need to execute a sort with a large number of options.

ISSORT uses the isCOBOL framework, so any configuration on it is inherited by ISSORT.

One of the following exit status is returned:

0	Success
2	Unsupported feature
15	Command statement error(s) detected
100	I/O error

Instructions

The following instructions are accepted by ISSORT.

- SORT / MERGE

These instructions specify either a sort or a merge option and must be followed by a FIELDS instruction specifying the field(s) to be used. Only one between these two options can be used.

- FIELDS

FIELDS allows you to specify the fields on which the file is sorted/merged; the syntax for this instruction is:

```
FIELDS (<start>,<length>,<type>,<order>[,<start>,<length>,<type>,<order>]...)
```

Where:

- *start* is the offset in bytes of the field starting from 1
- *length* is the length in byes of the field
- *type* is a two letters code indicating the format of the data. Allowed codes are:

bi	PIC 9 COMP
c5	PIC 9 COMP-5
c6	PIC 9 COMP-6
ch	Alphanumeric
cx	COMP-X
fl	floating point (length must be 4 or 8)
li	PIC S9 DISPLAY LEADING
ls	PIC S9 DISPLAY LEADING SEPARATE
nu	PIC 9 DISPLAY
pd	PIC S9 COMP-3

s5	PIC S9 COMP-5
sb	PIC S9 COMP
ti/zd	PIC S9 DISPLAY TRAILING INCLUDED
ts	PIC S9 DISPLAY TRAILING SEPARATE

- *order* is a one letter code indicating the ordering of the field in the output file. Allowed codes are:

a	ascending
d	descending

- USE / GIVE

These instructions allow you to specify the input file and output file, respectively, of a sort or merge process. The input and output file descriptions may include ORG, RECORD, and KEY phrases, which define the file's characteristics. The syntax for these instructions is as follows:

```
USE <filename>
                ORG <filetype>
                RECORD { F <r-length>             }
                       { V <r-length> <r-max-len> }
                KEY (<k-start>,<k-length>,<key-type>[,<k-start>,<k-length>,<key-type>].
..)


GIVE <filename>
                ORG <filetype>
                RECORD { F <r-length>             }
                       { V <r-length> <r-max-len> }
                KEY (<k-start>,<k-length>,<key-type>[,<k-start>,<k-length>,<key-type>].
..)
```

Where:

- *filename* is name of the file: it will be searched for according to the rules specified for the isCOBOL framework
- *filetype* is a two letter code indicating the format of the file. Allowed codes are:

ix	indexed file
ls	line sequential file
rl	relative file
sq	sequential file

- *r-length* is the length in bytes of the record
- *r-max-len* is the maximum length in bytes of the record
- *k-start* is the offset in bytes of the key, starting from 1
- *k-length* is the length in bytes of the key

- *key-type* is a code indicating the key type. Allowed codes are:

p	primary key
a	alternate key
ad	alternate key with duplicates
c	segment of the previous key

Note - The primary key must be specified before any other key.

If *filetype* is not specified then ORG=sq is implied.

RECORD F means that the record length is fixed while RECORD V means that the record length is variable. In the case of RECORD V, you must supply the maximum record length.

Note - the KEY instruction must be specified only when indexed files are involved.

- INCLUDE / OMIT

These instructions allow to specify a condition under which records may be included or excluded from, respectively, the output file. Only one of these instructions may be specified. Note that the comma characters can always be omitted in order to improve the readability of the expression.

The syntax for these instructions is as follows:

```
INCLUDE [<format-clause>] <cond-clause> [<format-clause>]
OMIT    [<format-clause>] <cond-clause> [<format-clause>]
```

Where:

- *Format-clause* has the following syntax:

```
FORMAT [=] <type>
```

Where *type* is a two letter code indicating the format of the data, as specified in the FIELDS instruction. This clause allows you to set a default data type to be used when no explicit data type is indicated. If this clause is omitted then its default value is 'nu'.

- *Cond-clause* has the following syntax:

```
COND [=] (<left-value>, <comp-op>, <right-value>
                              [,<log-op>, <left-value>, <comp-op>, <right-value>]...)
```

Where *left-value* is the combination of start, length and type with the same format and meaning as for the FIELDS instruction except that a further type 'ss' can be used: in such a case a substring search is performed, as explained below.
comp-op is one of the following two letters codes:

eq	equal to
ne	not equal to

gt	greater than
lt	less than
ge	equal to or greater than
le	equal to or less than

right-value can be either

- a combination of start, length and type as for left-value
- a constant string preceded by the letter c (e.g. c'some data')
- a constant number

log-op is one of the following codes:

and	and
&	and
or	or
|	or

The logical operators AND and OR are evaluated following the common precedence rule (AND is evaluated before OR), however the evaluation order can be altered by inserting parenthesis in the appropriate positions.
When the 'ss' type is used in a *left-value* then the following restrictions apply:

- *comp-op* must be either 'eq' or 'ne';
- *right-value* must be a constant string.

In such a case a substring search is performed: if *left-value* is bigger in size than *right-value* then the record is checked for an occurrence of what specified in *right-value*, otherwise if *right-value* is bigger in size than *left-value* then the string specified as *right-value* will be searched for an occurrence of the content of the record.
For example:

```
cond=15,3,ss,eq,c'J69L92'
```

is true if the record, starting at position 15 for 3 bytes, contains any of the following sequence: J69,69L,9L9,L92;

```
cond=1,10,ss,eq,c'J82'
```

is true if the record, starting at position 1 for 10 bytes, contains the sequence J82 in any position.

Syntax rules

All the keywords are case-insensitive.

Constant strings within the instructions can be enclosed in quote or double quote: in order to use the quotes in a string you must double them: for example if you want to process a file named *my'file* you can use one of the following expressions:

```
"my'file"
'my''file'
```

Example

The below commands sorts an indexed file and stores the sorted records in a line sequential text file.

The input file *idxfile* is an indexed file with a primary key with 2 segments and an alternate key with duplicates. Its record length is 40 bytes and it is fixed.

The output file *output.txt* is a line sequential file.

The sort is performed on the first 6 characters in descending order. Only the records containing a number, in unsigned display format starting at offset 37 and 4 bytes long, whose value is greater or equal to 902 are included in the output file.

```
issort take sort.cmd
```

Content of *sort.cmd*:

```
sort
 fields (1, 6, ch, d)
 use idxfile
     org ix record f 40 key (1, 6, p, 7, 15, c, 22, 15, ad)
 give output.txt
     org ls record f 40
 include cond = 37,4,ge,902
```

Thin Client

ISSORT can't be launched directly by the isCOBOL Client. If you need to perform a sort in thin client environment, either on the client machine or the server machine, you should consider calling the C$SORT routine.

ISUPDATER (Update Facility)

Overview

isCOBOL implements a software update feature that allows you to download updates through the HTTP and HTTPS protocols. It's a general-purpose updater that can be used to download any type of file, not only COBOL programs.

The update process

The isupdater tool connects to a given HTTP server and checks if a new version of the files is available. If a new version is found, the isupdater downloads the files. The HTTP server (server side) and isupdater (client side) are configured through property files. Files are stored in ZIP archives on the server and are automatically unzipped once downloaded to the client.

isCOBOL Server as an HTTP server

The isCOBOL Server can work as an HTTP server for isupdater, so you don't have to set up an HTTP server in order to take advantage of the software update feature.

Post update operations

At the end of the update process, isupdater can automatically run a program that takes care of post update operations such as copying the downloaded files in the proper folder. This program can be a standard COBOL program or a Java class executable from the command line (e.g. a Java class with a main() method).

Post update launch

Isupdater can automatically run a given class at the end of the update process. This class must be accessible in the Classpath.

ISUPDATER in the Thin Client environment

The isCOBOL Client automatically invokes the ISUPDATER utility in order to have libraries that are up to date with the remote isCOBOL Server. This process is explained in the Automatic Client update chapter.

ISUPDATER in the runtime environment

The isCOBOL Runtime invokes the ISUPDATER utility when launched with the -update option.

Server configuration

On the server machine, the folder that isupdater will check through HTTP must contain the following items:

1. A configuration file named *swupdater.properties*
2. A zip file or directory for every package described in the *swupdater.properties* file

See Server Configuration (swupdater.properties) for details about the properties that can be used in *swupdater.properties*.

The following example describes the update of isCOBOL's bin and lib folders to version 100.

Directory content:

```
isCOBOL-bin.zip
isCOBOL-lib.zip
swupdater.properties
```

Content of zip files:

isCOBOL-bin.zip	Native libraries typically installed in the isCOBOL's *bin* folder on Windows. The equivalent Unix libraries are installed in the isCOBOL's *native/lib folder* on Unix platforms. It's good practice to include only native libraries in the update process as executable files never change between two isCOBOL releases and they could be locked during the update process.
isCOBOL-lib.zip	Jar libraries typically installed in the isCOBOL's *lib* folder.

Content of *swupdater.properties*:

```
swupdater.version.isc_lib=100
swupdater.lib.isc_lib=isCOBOL-lib.zip

swupdater.version.isc_bin=100
swupdater.lib.isc_bin=isCOBOL-bin.zip
```

Note - whenever you edit the swupdater.properties file there's no need to restart the isCOBOL Server. The changes to this file will be automatically considered at the next update request.

isCOBOL Server as an HTTP server

In the directory where the zip files and *swupdater.properties* are stored you can start the isCOBOL Server as follows:

```
iscserver -hs [-hsport port] [-hsroot folder]
```

Where

- *port* specifies the port used for the HTTP connection. If omitted, 10996 is used.
- *folder* specifies the root folder where *swupdater.properties* is located. This folder is also used to resolve relative paths of files in *swupdater.properties*. If omitted, the iscserver working directory is used.

The activation of the HTTP server as well as *port* and *folder* can also be set in the configuration file; see iscobol.as.httpserver (boolean), iscobol.as.httpserver.port and iscobol.as.httpserver.root for details.

The -hs option can be used along with other iscserver options. The following command starts the isCOBOL Server with all the services on: application server, file server and HTTP server.

```
iscserver -as -fs -hs
```

Usage and configuration of isCOBOL Updater

Once the HTTP server has been properly configured, it's possible to run the isupdater tool from the client machines.

Usage

```
iscupdater -c configuration_file [-stop] [-icon image_file]
```

Where:

- *configuration_file* is the properties file used to pass the server location and current version to the isupdater tool.
- *-stop*, if used, the isupdater tool just checks for the availability of updates and, if one is available, it informs the user with a message box, but nothing is downloaded.
- *image_file* is a picture suitable for Java (e.g. a GIF or PNG file) that will be shown as custom icon instead of the Java logo.

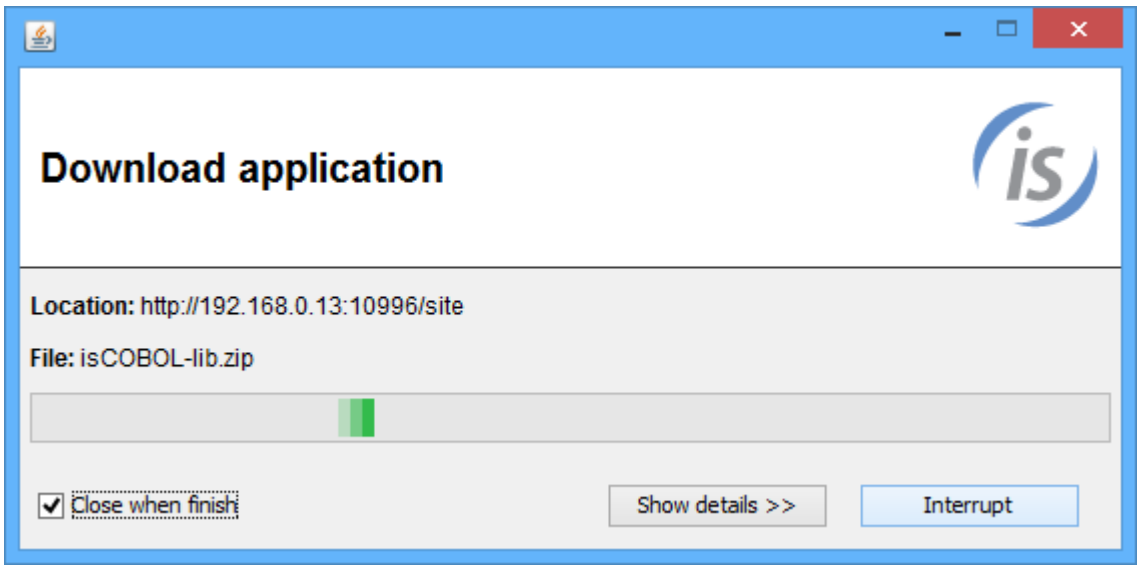
Client configuration

See Client Configuration (isupdater.properties) for the list of the properties that can be used to configure the isupdater tool.

The following sample configuration sets the HTTP server address for updates to 192.168.0.13, listening on the port 10996. The swupdater.properties file is expected in the sub directory "site". It also sets the current version of isCOBOL's bin and lib folders to 90 and the folder where downloaded files should be saved to C:\Veryant:

```
swupdater.site=http://192.168.0.13:10996/site
swupdater.version.isc_lib=90
swupdater.version.isc_bin=90
swupdater.directory.isc_lib=C:/Veryant/lib
swupdater.directory.isc_bin=C:/Veryant/bin
swupdater.mainclass=com.iscobol.invoke.Isrun -v
```

Since the client side version (90) is less that the version available on the server (100, as referenced in the server configuration file explained on the previous page), the download process starts.



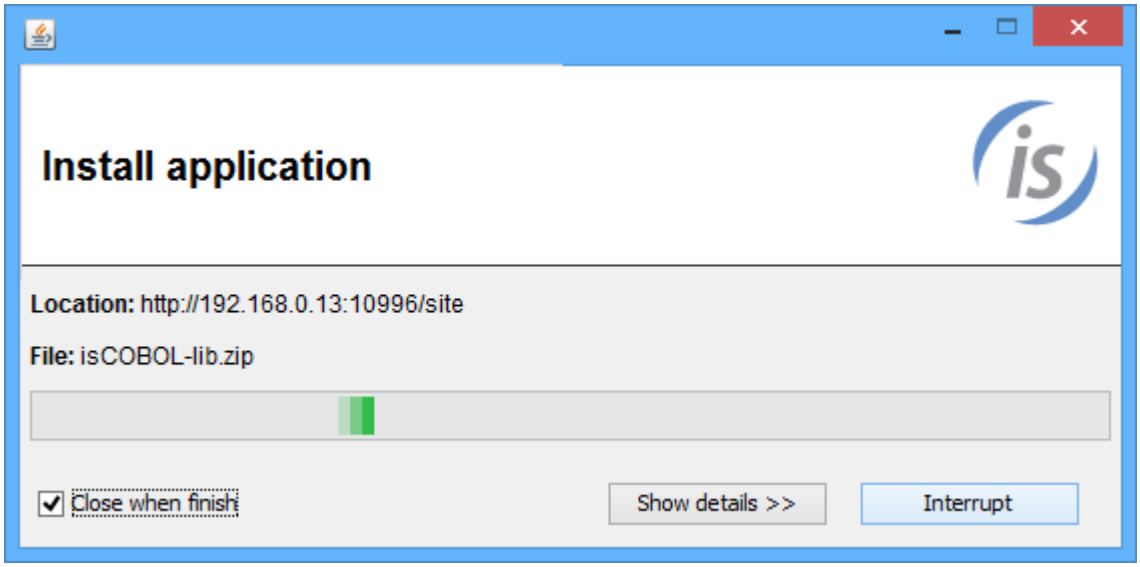
At the end of the download process, the isupdater tool rewrites the configuration file putting the new version in it:

```
#Automatically updated
swupdater.site=http://192.168.0.13:10996/site
swupdater.version.isc_lib=100
swupdater.version.isc_bin=100
swupdater.directory.isc_lib=C:/Veryant/lib
swupdater.directory.isc_bin=C:/Veryant/bin
swupdater.mainclass=com.iscobol.invoke.Isrun -v
```

Post update

Once the download process has been completed, the folders isCOBOL-lib and isCOBOL-bin will be available in C:\Temp. If they contain a class named POSTUPDATE.class, then isupdater will automatically run it passing the destination folder as a parameter. You can use this feature to code post update operations such as copying downloaded files to a different folder.

During the execution of this class, the isupdater dialog changes as follows:



If the property swupdater.mainclass was set, then isupdater automatically runs the specified class along with the arguments. In the above example the command Isrun -v is executed in order to show a message box with the current version of the runtime.

The following alternate example shows how to call a server program named MENU using the thin client technology after the update:

```
swupdater.mainclass=com.iscobol.gui.client.Client MENU
```

Setup of an update server for the isCOBOL SDK

In this chapter we explain the good practice setup an update server that can be used by both the -update option of iscrun and the Automatic Client update to update their runtime component before starting.

Our update server will be able to serve different client platforms so we may take particular attention to the native items included in the isCOBOL SDK. Most of the isCOBOL Framework components are pure Java programs and therefore they're cross-platform. However there are a couple of components that are (or include) native platform dependent items. These items are:

- native libraries like iscobolc (bin\iscobolc.dll on Windows, /native/lib/libiscobolc.so on Linux/Unix and /native/lib/libiscobolc.jnilib on MacOSX),
- SWT libraries for the web-browser implementation installed in the lib folder.

The suggested steps are:

1. Create a folder that will host the files for the update.

2. Within the folder just created, create the following sub folders and fill them as suggested:

Folder	What to put inside
lib	All the jars found in the isCOBOL lib folder except for swt-<platform>.<arch>.jar You can take these libraries from any isCOBOL distribution as they're cross-platform. If your application calls programs on the client side via CALL CLIENT in a Thin Client environment, collect the classes of these programs in a jar library and add the library to this lib folder.
libWin32	swt-Windows.32.jar taken from the isCOBOL installation for Windows 32 bit
libWin64	swt-Windows.64.jar taken from the isCOBOL installation for Windows 64 bit
libLinux32	swt-Linux.32.i586.jar taken from the isCOBOL tar.gz for Linux 32 bit
libLinux64	swt-Linux.64.x86_64.jar taken from the isCOBOL tar.gz for Linux 64 bit
libMac64	swt-MacOSX.64.x86_64.jar taken from the isCOBOL tar.gz for Mac OSX 64 bit
nativeWin32	All the DLL files found in the bin directory of the isCOBOL installation for Windows 32 bit
nativeWin64	All the DLL files found in the bin directory of the isCOBOL installation for Windows 64 bit
nativeLinux32	All the so files found in the native/lib directory of the isCOBOL tar.gz for Linux 32 bit
nativeLinux64	All the so files found in the native/lib directory of the isCOBOL tar.gz for Linux 64 bit
nativeMac64	All the jnilib files found in the native/lib directory of the isCOBOL tar.gz for Mac OSX 64 bit

At the end you should have the following situation:

Folder	Content
lib	asm-7.2.jar asm-commons-7.2.jar asm-tree-7.2.jar bcprov-jdk14-1.38.jar charva.jar commons-codec-1.13.jar commons-collections4.4.4.ja commons-compress-1.19.jar commons-math3-3.6.1.jar commons-logging-api.jar commons-logging.jar ctree-rtg.jar DJNativeSwing-SWT.jar DJNativeSwing.jar image4j-0.7.2.jar iscobol.jar isprofiler.jar isupdater.jar itext-2.1.7v5.jar jacoco-core-0.8.5.jar javassist.jar jcommon-1.0.23.jar jcommon-xml-1.0.23.jar jcalendar-1.3.3.jar jcommon-1.0.23.jar jcommon-xml-1.0.23.jar jdom.jar jfreechart-1.5.1.jar jna-5.2.0.jar jna-platform-5.2.0.jar joe-1.3.jar poi-4.1.2.jar poi-ooxml-4.1.2.jar poi-ooxml-schemas-4.1.2.jar swt-Windows.##.jar utility.jar wow.jar wowax.jar xmlbeans-3.1.0.jar ... and optionally the jar library that hosts programs called via CALL CLIENT.
libWin32	swt-Windows.32.jar
libWin64	swt-Windows.64.jar
libLinux32	swt-Linux.32.i586.jar
libLinux64	swt-Linux.64.x86_64.jar
libMac64	swt-MacOSX.64.x86_64.jar

Folder	Content
nativeWin32	ctree.dll dyncall.dll dyncall_n.dll iscobolc.dll iscobolc_n.dll msvcm90.dll msvcp90.dll msvcr90.dll Terminal.dll
nativeWin64	ctree.dll dyncall.dll dyncall_n.dll iscobolc.dll iscobolc_n.dll msvcm90.dll msvcp90.dll msvcr90.dll
nativeLinux32	libctree.so libdyncall.so libdyncall_n.so libiscobolc.so libiscobolc_n.so libTerminal.so
nativeLinux64	libctree.so libdyncall.so libdyncall_n.so libiscobolc.so libiscobolc_n.so libTerminal.so
nativeMac64	libctree.jnilib libiscobolc.jnilib libiscobolc_n.jnilib libTerminal.jnilib

3. Create a file named *swupdater.properties* in the same directory of the above folders and fill it as follows:

```
swupdater.version.iscobol=1011
swupdater.lib.iscobol=lib
swupdater.lib.win.32.iscobol=libWin32
swupdater.lib.win.64.iscobol=libWin64
swupdater.lib.linux.32.iscobol=libLinux32
swupdater.lib.linux.64.iscobol=libLinux64
swupdater.lib.mac.64.iscobol=libMac64
swupdater.version.iscobolNative=1011
swupdater.lib.win.32.iscobolNative=nativeWin32
swupdater.lib.win.64.iscobolNative=nativeWin64
swupdater.lib.linux.32.iscobolNative=nativeLinux32
swupdater.lib.linux.64.iscobolNative=nativeLinux64
swupdater.lib.mac.64.iscobolNative=nativeMac64
```

Note - the above snippet assumes that you're providing items from isCOBOL 2020 R1 build 1011. If you're providing another build of isCOBOL, replace 1011 with the correct build number.

The server configuration is ready, now you can start an isCOBOL Server with the -hs option as described in isCOBOL Server as an HTTP server. The isCOBOL server must point to the directory created at step 1; in order to achieve it, you can either start the isCOBOL Server from that directory or use the -hsroot option to point to that directory.

Alternatively, instead of using isCOBOL Server, you can install the directory created at step 1 into your favorite HTTP server (e.g. Apache or IIS).

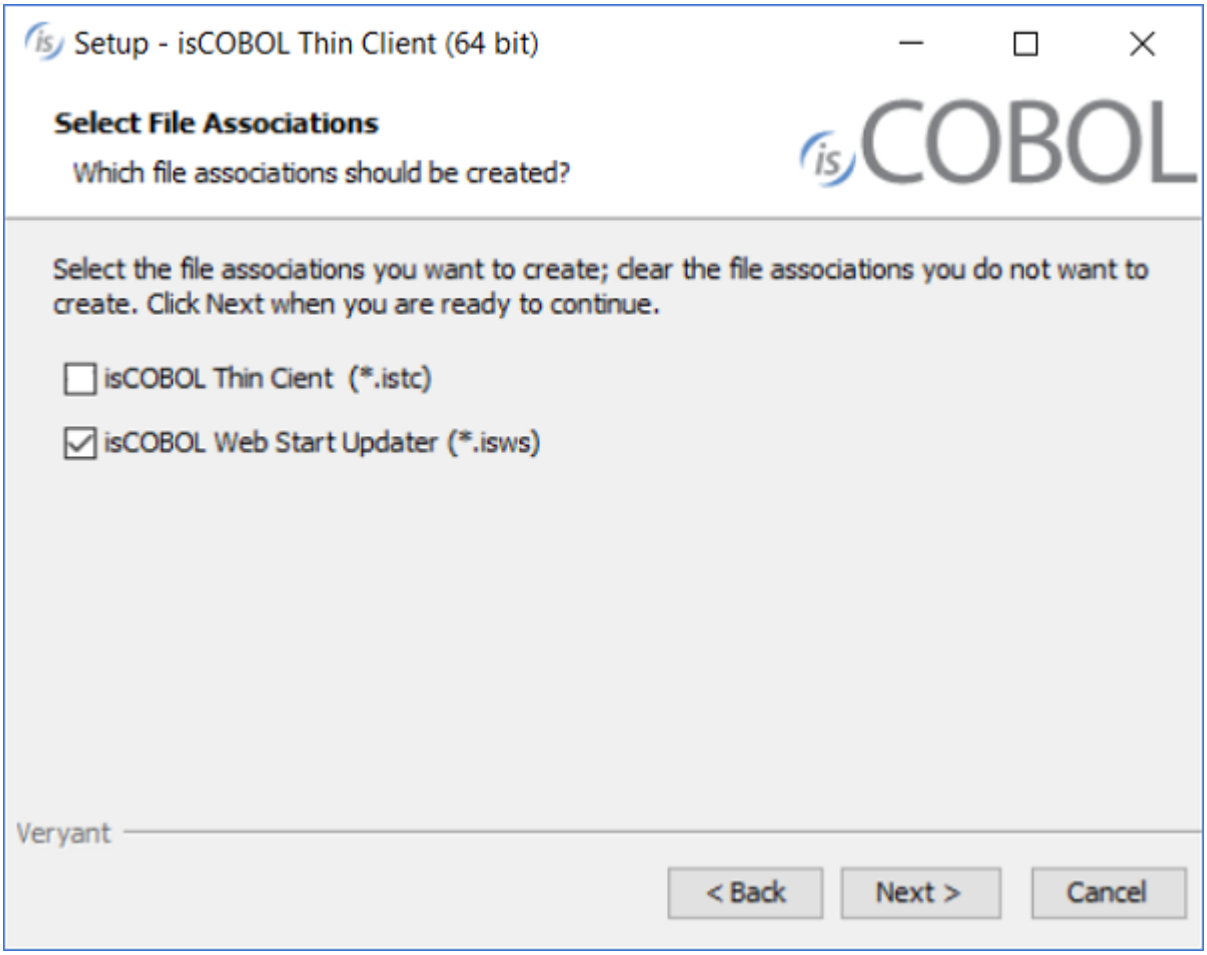
Distribution of a COBOL application through the Update Facility and isws files

The information in this chapter is applicable to thin client environments where the client machines are Windows.

To deploy the application through the Update Facility of isCOBOL Server, the isCOBOL Client must be installed on the client machines.

Install either isCOBOL_*yyyy*_*R*_*n*_Windows_*arc*.exe (it requires Java installed on the machine) or isCOBOL_*yyyy*_*R*_*n*_THIN_Windows_*arc.ext* (it doesn't require Java on the machine as it installs its own JVM) where *yyyy* is the year, *R* is the release number, *n* is the build number, *arc* is the system architecture and *ext* is either "exe" or "msi".

When prompted, choose to associate the isws extension to the isCOBOL Client:



The isws files are property files that include the isUpdater configuration. See Client Configuration (isupdater.properties) for the list of properties that you could include in this kind of file. The isws files could be passed to isUpdater via the -c option. The installer creates an association between the isws extension and the command:

```
isupdater -c %1
```

Below we describe how to set up the deployment of a COBOL application through the Update Facility and isws files, using the isCOBOL Server as HTTP server for the download of the application.

What to do server side

The user should gather the platform-dependent components from the corresponding isCOBOL setups and place them in specific folders on the server machine. Also the COBOL application items (classes, configuration and icons) should be gathered in a dedicated folder.

Here is a suggestion: create the following subfolders in the isCOBOL installation directory (that we assume as */home/veryant/isCOBOL2020R2*):

Directory	What to copy inside
libWin32	Content of the lib folder of isCOBOL for Windows 32 bit

Directory	What to copy inside
libWin64	Content of the lib folder of isCOBOL for Windows 64 bit
binWin32	Content of the bin folder of isCOBOL for Windows 32 bit
binWin64	Content of the bin folder of isCOBOL for Windows 64 bit
myApp	Items of the COBOL application, described later

The isCOBOL Server must be started with the option –hs in order to activate the HTTP Server feature, e.g.

```
iscserver -hs
```

Create a file named *swupdater.properties* in the isCOBOL Server's working directory and put the following entries into it:

```
swupdater.version.iscobol=###
swupdater.lib.win.32.iscobol=/home/veryant/isCOBOL2020R2/libWin32
swupdater.lib.win.64.iscobol=/home/veryant/isCOBOL2020R2/libWin64
swupdater.version.iscobolNative=###
swupdater.lib.win.32.iscobolNative=/home/veryant/isCOBOL2020R2/binWin32
swupdater.lib.win.64.iscobolNative=/home/veryant/isCOBOL2020R2/binWin64
swupdater.version.myApp=1
swupdater.lib.myApp=/home/veryant/isCOBOL2020R2/myApp
```

Where ### is the build number of the isCOBOL Server. For example, for "release 2020 R2 build#1023.5-20200911-30388" you would use "1023.5".

If you use third party jar libraries that need to be installed along with your COBOL application, copy them to the isCOBOL *lib* folder.

Put the following items in the *myApp* subfolder:

- The class files of your COBOL programs,
- The icons (bmp, jpeg, gif and png) used by your programs,
- A file named *myApp.properties* that contains
    - a valid runtime license,
    - a code-prefix setting that points the folder C:\myApp (e.g. iscobol.code_prefix=C:\\myApp),
    - the configuration of your COBOL application (e.g. keystrokes and file-prefix).

If you have custom native libraries that should be installed along with your application, copy them to the proper "bin<Platform>" folder (e.g. if you have a library named *mylib.dll* for both Windows 32 bit and Windows 64 bit, copy the 32 bit version to *binWin32* and copy the 64 bit version to *binWin64*).

What to do client side

Create a file with isws extension, e.g. *myapp.isws*, and put the following entries into it:

```
swupdater.site=http://serverNameOrIp:10996
swupdater.version.iscobol=###
swupdater.directory.iscobol=C:/Users/UserName/Veryant/isCOBOL THIN2020R2/lib
swupdater.directory.clean.iscobol=true
swupdater.version.iscobolNative=###
swupdater.directory.iscobolNative=C:/Users/UserName/Veryant/isCOBOL THIN2020R2/bin
swupdater.directory.clean.iscobolNative=true
swupdater.version.myApp=0
swupdater.directory.myApp=C:/myApp
swupdater.directory.clean.myApp=true
swupdater.mainclass=com.iscobol.invoke.Isrun -c C:/myApp/myApp.properties MYPROG
```

Where ### is the build number of the runtime installed by isCOBOL THIN. For example, for "release 2020 R2 build#1023.4-20200827-30295" you would use "1023.4".

MYPROG is the name of the program that you wish to execute. The class of this program must be found in the myApp folder discussed above.

The above snippet assumes that isCOBOL THIN was installed in the default location proposed by the setup wizard.

Note that *swupdater.version.myApp* in this file has a lower value of the corresponding property in the *swupdater.properties* file on the server. This is necessary to trigger the download of the COBOL Application (*myApp*) from the server machine to the local machine. After the first launch, *swupdater.version.myApp* is updated and its value matches the value server side. If you change the content of the *myApp* folder on the server, increase the value of *swupdater.version.myApp* in *swupdater.properties* on the server machine to trigger a new download of the *myApp* folder.

Double clicking on *myapp.isws* will trigger the program execution. It also will update the local copy of the isCOBOL runtime if necessary.

The file myapp.isws could be distributed via internet in the form of a file to be downloaded and executed.

JDBC2FD

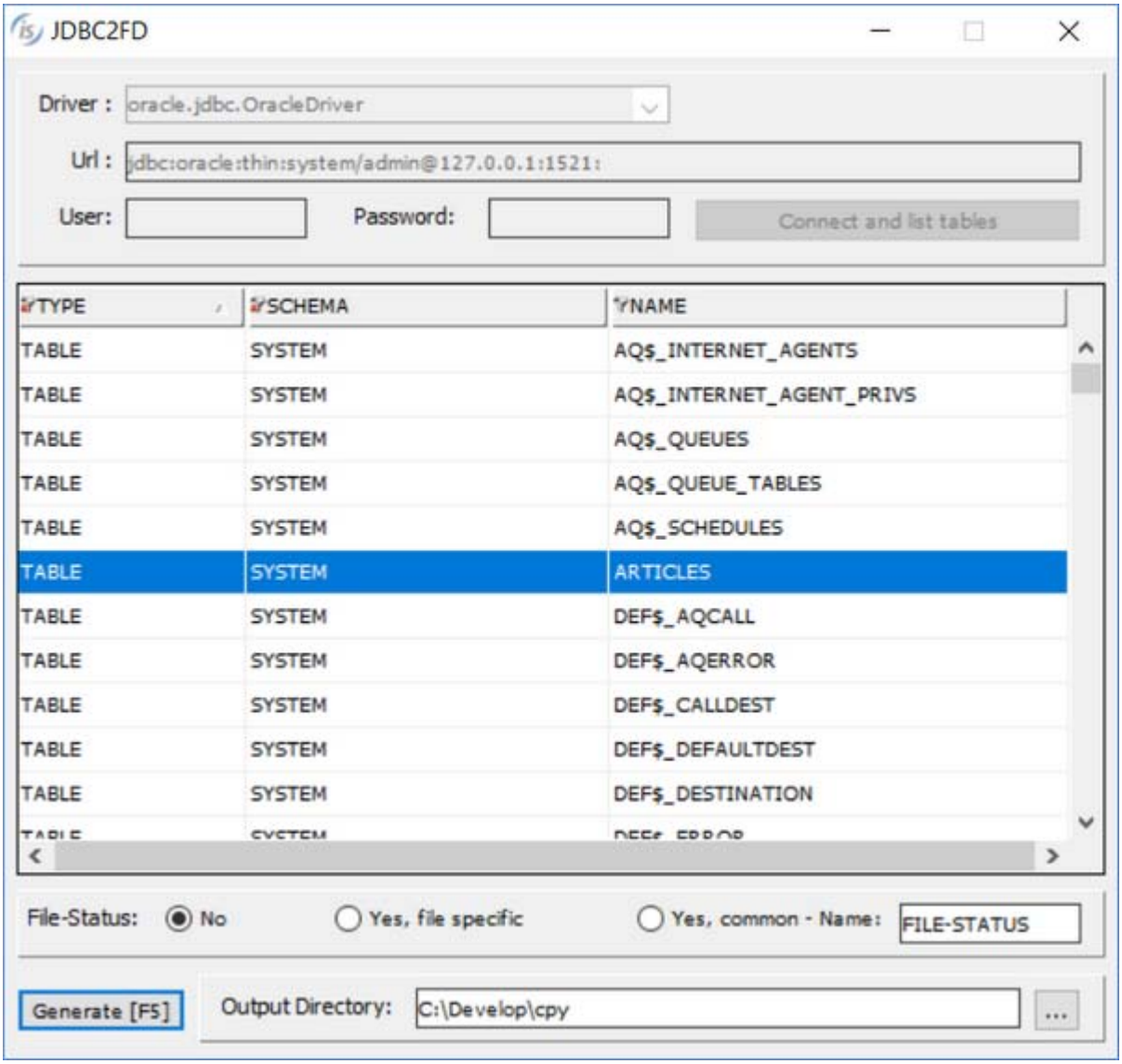
The JDBC2FD utility generates file description copybooks from database tables.

GUI Usage:

```
jdbc2fd
```

or

```
iscrun [-c jdbc2fd.properties] -utility jdbc2fd
```



Select the JDBC Driver (or type it if it doesn't appear in the list) and type the JDBC Url, then click "Connect and list tables" to obtain the list of the tables.

The Url field contains the base url while User and Password allow you to specify the username and password respectively. You can compile all the fields or provide a full url in the first and leave the other two blank.

Results can be filtered using the filters on column headings or pressing Ctrl+F.

The File-Status generation is customizable. You can choose to not generate any File-Status in the sl copybook or you can choose to generate a File-Status with the file name as prefix or a File-Status with a given name.

Choose the output directory for the copyfiles that will be generated, select the desired table and click the "Generate" button or press F5 to generate copybooks.

Command-line Usage:

```
jdbc2fd tableName [outputDirectory]
```

or

```
iscrun [-c jdbc2fd.properties] -utility jdbc2fd tableName [outputDirectory]
```

JDBC settings are retrieved by the iscobol.jdbc.driver and iscobol.jdbc.url configuration properties. If outputDirectory is omitted, the current directory is used.

Three copybooks are generated for each table.

tableName.sl	SELECT statement for FILE-CONTROL paragraph.
tableName.fd	File description entry and include of the below copybook
tableName.wrk	Record description

Thin Client

JDBC2FD can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility jdbc2fd <arguments>
```

The JDBC driver must be available in the server Classpath and copy files are generated on the server machine.

JOE

isCOBOL programs can be distributed from a server with isCOBOL Application Server; this is the architecture of choice for multiuser applications due to its many benefits. However, in order to get the best results all the processing needs to be in the isCOBOL environment.

Many legacy applications are made intermixing COBOL programs and interpreted scripts of some kind (e.g. Bourne shell). Until now these scripts needed to be rewritten into COBOL programs in order to run in isCOBOL Application Server.

This process can be lengthy and the results unsatisfactory because:

- interpreted procedures are now compiled procedures;
- procedures written with a language oriented to manage operating system tasks are now written using COBOL (a Business Oriented Language).

So, in order to speed up the migration process getting at the same time a better result, it would be useful to have a scripting language whose features are:

- ability to access any isCOBOL/Java resource: isCOBOL (as well as Java) is operating system independent therefore the Java environment is its virtual operating system;
- easy to change in order to get similar to any script language in terms of capability and readability;
- easy to customize in order to get frequently used operations at hand;

- easy to extend in order to be useful also for future applications' enhancements, not only for the migration process;
- easy to understand and use;
- 100% compatible with the isCOBOL Application Server architecture.

The Java Objects Executor (JOE for short) complies with the above requests.

JOE's only task is to execute methods of Java/isCOBOL objects in sequence on the fly: how it can be used to mimic any scripting language it will become clearer later.

Getting Started

JOE is installed along with isCOBOL.

You can run it interactively with the following command:

```
iscrun -joe
```

The shell is started and waits for commands:

```
CobShell interactive ready, type 'exit' to exit the session

cs>
```

In order to run a script you must supply the script name as argument, e.g.

```
iscrun -joe myscript.joe
```

JOE scripts can also be edited and executed in the isCOBOL IDE.

To edit a script within the IDE, add it to your project and open it with the JOE Editor.

To run a script from the IDE, right click on the script name in the File View and choose *Run As* > *Joe Application*. The output is shown in the Console view.

Basic syntax

In order to invoke methods we need an object oriented syntax: the obvious choice could be the COBOL OO syntax or, as an alternative, the Java syntax. We choose instead to use a syntax close to Smalltalk since it has a better readability, especially when used extensively. The JOE syntax for invoking a method is:

```
[ variable-name := ] object [method-name { ; | argument[,argument…]} …] .
```

In JOE any data is an object and any invocation is supposed to return an object. You can use variables to hold objects; a variable doesn't need to be declared and has no type, it can contain any type of object.

JOE has not reserved words, but it has few reserved symbols:

:=	assignment
;	no argument
,	parameter separator

	.	end of the message

The basic core of JOE is the triplet object-method_name-argument. If you want invoke a method with no argument then this must be explicitly stated in the code: the meaning of the semicolon character is exactly this, it means 'there are no arguments'.

Let's see some examples of this syntax compared with the equivalent Java syntax in order to get a better understanding.

JOE syntax	Java syntax	
A B C.	A.B(C);	Basic method invocation, A is an object, B is a method name and C is an object passed to the method as argument.
A B C D E.	A.B(C).D(E);	A is an object, B is a method name and C is an object passed to the method as argument. This invocation returns an object, so D is a method name and E its argument and so on. There is no theoretical limit to the length of a message.
A B; D E.	A.B().D(E);	This case differs from the previous one in that no argument is passed to method B.
A B C,D,E.	A.B(C,D,E);	In this case instead 3 arguments are passed to method B.
A B (C D E).	A.B(C.D(E));	The evaluation order can be altered using the parentheses (), which allow to execute a method and use its result as argument of another method. In this case the method D of object C is executed and its result is passed as argument to method B of object A.

You can assign the result of the last invocation in a message to a variable, e.g.:

```
var := A B C D E.
```

JOE supports two types of comments:

- in-line comments. JOE ignores everything from *> to the end of the line, e.g.:

```
*> this is a comment
```

- multi-line comments, JOE ignores everything from /* to */, e.g.:

```
/*
 This is a comment
 distributed on
 multiple lines
*/
```

Moreover a line starting with the sequence #! is ignored in order to support the shebang interpreter directive of the Unix-like operating systems.

Since there are no built-in instructions, in order to run something JOE needs an object that act as starting point. This object, let's call it 'command', is automatically loaded at the beginning of the execution. Since the command object is supposed to have useful and often used methods, it has been named '!' (the character for exclamation mark or bang) so that you need to type only one character and it is easy to see it in the source code.

The key point is that this command object has nothing special, it is a plain isCOBOL/Java object accessed using the Java reflection: you can write your own version if you like, inheriting the behaviors from the supplied one or even creating a brand new environment.

You can find a list of the currently available methods in Javadoc documentation.

We are now ready to do the very first program using JOE. We are going to use CobShell, a version of JOE that has been made similar to COBOL. The string "cs> " is the prompt and it is not part of the commands.

So the first program is the classic "Hello".

```
cs> ! display "Hello #1".
Hello #1
cs>
```

You can see the triplet object-method_name-argument very clearly here. The COBOL syntax would be *invoke command "display" using "Hello #1"* while the Java syntax would be *command.display("Hello #1")*. Since the bang cannot be used in a JOE name, you can also write:

```
cs> !display "Hello #1".
Hello #1
cs>
```

The meaning is the same as the previous one.

The method *display* accepts any number of parameters, shows them, issue a new line and returns the command object itself. If the method is invoked without parameters, only a new line is issued. There is an equivalent method, *displayNoAdv*, that does the same things without issuing a new line.

So you can get the same result with the following line:

```
cs>!display "Hello #",1.
Hello #1
cs>
```

More than one invocation can be concatenated: when it happens the first triplet is executed and the result is the object of a second triplet and so on until the line is closed. The dot character used to stop the evaluation. So you can issue the following line:

```
cs> !display "Hello #",1 display "Hello #",2.
Hello #1
Hello #2
cs>
```

Note that the second *display* doesn't need the bang since the command object is returned by the first one. Another example is:

```
cs> ! display "Hello #",1 display; display "Hello #",2.
Hello #1

Hello #2
cs>
```

Note that the second *display* is followed by a semicolon in order to inform the interpreter that that method has no arguments.

The evaluation is done left to right but you can change the evaluation order by enclosing the object expression to evaluate before between parenthesis. In order to see that, it comes handy to know that a literal string is an object itself and it is equivalent to the Java String object, therefore it has the method *length* that returns the length of the string. So you can issue the following line:

```
cs> !display "Length",("Length" length;).
Length6
cs>
```

The *length* method is executed before the *display* method and the result is used as parameter by it.

In this case you can avoid the use of the semicolon since the parameters of a method cannot be placed after a closed parenthesis.

Variables and literals

JOE allows you to store an object reference in a variable through the symbol :=. A variable name consists in a sequence of characters that are not reserved for other uses, as space, ! etc. (the exact set is still to be defined, the Java names will be valid for sure). The variables are not typed, so you can use them for any kind of object. They can also change type during the execution. For example:

```
cs> a := "Length".
cs> !display a,(a length).
Length6
cs> a := !.
cs> a display "Hello!".
Hello!
cs> a := "Length" length.
cs> ! display a.
6
cs>
```

If a variable is used without any previous assignment, its value will be null.

```
cs> !display b.
(null)
cs>
```

Currently JOE manages only three types of literals, integer numbers, floating point double precision numbers and strings. They are objects equivalent to java.lang.Integer, java.lang.Double and java lang.String but they are actually wrapped in internal objects in order to get more functionalities. For example the Java String allows you to easily see if two objects are equal through the method *equals*, however if you want compare two instances of String in order to know which is the greater, you need to use the method *compareTo* and then look for the result. The JOE wrapped objects all have the methods *gt*, *ge*, *lt*, *le*, *ne* that allow you to easily compare to literals of the same kind, e.g.:

```
cs> ! display ("A" gt "B").
false
cs> ! display ("A" lt "B").
true
cs>
```

Numbers are wrapped as well in order to get all the arithmetic operation at hand. JOE doesn't need the use of arithmetic operators characters nor logic operators characters, so they are automatically translated in words according the following list:

=	equals
<	lt
>	gt
<=	le
>=	ge
<>	ne
+	add
-	subtract
*	multiply
/	divide
%	mod

This translation allows you to issue invocations as the following:

```
cs> ! display ("A" > "B").
false
cs> ! display ("A" < "B").
true
cs> ! display (1 + 2).
3
cs> ! display (1 + 2 * 3).
9
cs> ! display (1 + (2 * 3)).
7
cs>
```

Note that the common arithmetic operations precedence is not respected, the evaluation is always left to right and if you want to change it you have to use the parenthesis.

Any time you use a literal to call an external object, it is converted into the correspondent Java object and the returned object is converted to the internal object when needed. In order to see that, you must know that the supplied command object has the method *newInstance* that allows you to instance any isCOBOL/Java object. The following example shows you how standard Java object can be handled by JOE:

```
cs> bd1 := !newInstance "java.math.BigDecimal","5.0".
cs> bd2 := !newInstance "java.math.BigDecimal",7.
cs> ! display "bd1=", bd1, "; bd1^2=", (bd1 pow 2).
bd1=5.0; bd1^2=25.00
cs> ! display "bd1 scale=",(bd1 scale).
bd1 scale=1
cs> ! display "bd1+bd2=", (bd1 + bd2).
bd1+bd2=12.0
cs>
```

You can also create your own objects and easily handle them in the JOE environment; let's say you want to handle dates in your procedures, you could write a Java class like the following one:

```
import java.util.Date;
public class MyDate {
   private static long msPerDay = 1000 * 60 * 60 * 24;
   private final java.util.Date date;
   public MyDate(long time) {
      date = new java.util.Date(time);
   }
   public MyDate(int year, int month, int day) {
      date = new java.util.Date(year - 1900, month - 1, day);
   }
   public long subtract (MyDate d) {
      return (date.getTime() - d.date.getTime()) / msPerDay;
   }
   public MyDate subtract (int days) {
      return new MyDate (date.getTime() - (days * msPerDay));
   }
   public MyDate add (int days) {
      return new MyDate (date.getTime() + (days * msPerDay));
   }
   public boolean equals (Object d) {
      if (d instanceof MyDate)
         return date.equals (((MyDate) d).date);
      else
         return false;
   }
   public boolean lt (MyDate d) {
      return date.before (d.date);
   }
   public boolean gt (MyDate d) {
      return date.after (d.date);
   }
   public String toString()  {
      return date.toString();
   }
}
```

After compiling this class and having it accessible through CLASSPATH, you can issue the following invocations:

```
cs> amRev := !newInstance "MyDate",1775,04,19.
cs> frRev := !newInstance "MyDate",1789,05,05.
cs> ! display "American revolution start=",amRev.
American revolution start=Wed Apr 19 00:00:00 CET 1775
cs> ! display "French revolution start=",frRev.
French revolution start=Tue May 05 00:00:00 CET 1789
cs> ! display "years between the revolutions=",((frRev - amRev) / 365).
years between the revolutions=14
cs> ! display (frRev > amRev).
true
cs> ! display (frRev < amRev).
false
cs> ! display (amRev = (!newInstance "MyDate",1775,04,19)).
true
cs> ! display "15 days after=",(amRev + 15).
15 days after=Thu May 04 00:00:00 CET 1775
cs> ! display "15 days before=",(amRev - 15).
15 days before=Tue Apr 04 00:00:00 CET 1775
cs>
```

JOE doesn't handle arrays however you can get the same behavior through the use of objects. For example the standard command implements the method *array* that returns the equivalent of a Java array containing the arguments as elements, e.g.:

```
cs> myArray := !array 1,"two",3.0.
cs> !display (myArray get 0).
1
cs> !display (myArray get 1).
two
cs> !display (myArray get 2).
3.0
cs>
```

The method *set index,value* allows you to set a value in an array. Since an element of an array can be any type of object, you can have elements that are arrays themselves, recreating the behavior of multidimensional arrays.

At this point you should see that, through the application of few simple rules, you can get an easy-to-use powerful environment customized on your needs. However, in order to get a complete language, it is necessary to have some decisional control structure. We need then to introduce a further concept, the Block.

Blocks

A block is simply a list of invocations enclosed between braces. It is an object itself so you can assign it to a variable, e.g.:

```
cs> a := { b := 2. ! display (b + 1). }.
cs>
```

The block content is not executed, it is only stored; since it is an object, in order to execute its content, you only need to invoke its method *exec*.

```
cs> a := { b := 2. ! display (b + 1). }.
cs> a exec.
3
cs>
```

The method *exec* of a block returns the result of the last invocation; in the case above it will return the result of the display, i.e. the command object.

```
cs> a := { b := 2. ! display (b + 1). }.
cs> a exec; display "end".
3
end
cs>
```

(Note the use of the semicolon character in order to inform the interpreter that exec has no parameters)

The blocks allow to easily implement a method that issue the behavior of an "if" statement: the following Java method is the implementation issued in the supplied command object:

```
public Object $if (Boolean cond, Block ifTrue) throws Exception {
   Object Return = cond;
   if (cond.booleanValue()) {
      Return = ifTrue.exec();
   }
   return Return;
}
```

You can note that the name of this method is *$if* : Java doesn't allow to have methods names equal to a reserved word, so when the interpreter recognizes a method name that equals a Java reserved word, it automatically prefixes the method name with the character "$".

Now you can issue an invocation like the following one:

```
cs> a:=1. b:=1. !if (a=b),{!display "a=b".}.
a=b
cs>
```

The "else" behavior can be achieved with a further method, similar to the previous one:

```
public Object $if (Boolean cond, Block ifTrue, Block ifFalse)
                                                    throws Exception {
   Object Return;
   if (cond.booleanValue()) {
      Return = ifTrue.exec();
   } else {
      Return = ifFalse.exec();
   }
   return Return;
}
```

As an example:

```
cs> a:=1. b:=2. !if (a=b),{!display "a=b".},{!display "a<>b".}.
a<>b
cs>
```

In the above example all the code is written on a single line, you can improve the readability writing it on multiple lines. CobShell can be executed with a text file name as parameter and in such a case the content of the file is executed. You can then write the above example in the following way:

```
a:=1.
b:=2.
!if (a=b),{
   !display "a=b".
},
{
   !display "a<>b".
}.
```

Blocks are used also to perform loops: the method *until* execute a block until the specified condition (included in a block) is true. For example:

```
cs> a:=0. !until { a=5 },{ a := a + 1. !display "a=",a. }. !display "end".
a=1
a=2
a=3
a=4
a=5
end
cs>
```

The condition must be included in a block because the condition must be re-evaluated at the beginning of each cycle. Since the execution of a block returns the result of the last invocation, the above example can also be written in the following way:

```
cs> a:=0. !until { a:=a+1. a>5 },{ !display "a=",a. }. !display "end".
a=1
a=2
a=3
a=4
a=5
end
cs>
```

(Note that in thes case the condition is a>5 instead of a=5 : this because the increment of the variable is issued before the evaluation of the condition instead of inside the second block).

At this point you have a complete language with all the necessary features. A subroutine can be implemented as a block, assigned to a variable and executed when needed.

The following example is a procedure that guesses a user thought number and summarizes what has been seen so far.

```
answer := "".
high := 1023.
low := 1.
ntry := 1.

!display "Think to a number between ",low," and ",high,
                                    ": I can guess it using 10 tries at most".

!until { answer = "c" },
{
   try := ((high - low) / 2 + low).
   !display "My guess is ", try.
   !display "Is the guess (c)orrect, too (h)igh or too (l)ow?".
   answer := !accept.
   !if (answer = "c"), {
      !display "I guessed the number using ",ntry," guesses".
   } , {
      !if (answer = "h"), {
         high := try.
         ntry := ntry + 1.
      } , {
         !if (answer = "l"), {
            low := try.
            ntry := ntry + 1.
         } , {
            !display "Answer with 'c', 'h' or 'l' please".
         }
      }
   }
}.
```

It is possible to achieve the behavior of more complex statements, like a multi-way branch similar to the COBOL EVALUATE statement.

The *evaluate* method takes an object as an argument and returns an object that has the method *when* that typically has two arguments, an object and a block:

if the argument is equal to the one specified in evaluate then executes the block, updates its state and returns itself in the event of further invocations of *when*.

The object used to implement this feature has its own internal state that allows the execution only of the first block that satisfies the condition.

The method *when* can also be invoked without specifying any block, in which case the condition of equality is still checked and put in OR with the next invocation of *when*.

The *when_other* method takes a block as an argument that runs only when no other block has been executed previously.

The method *end_evaluate* finally makes sure that the result of the last run is returned by the evaluate at the end of all the invocation.

Here is the previous example implemented by the using of the *evaluate* method:-

```
answer := "".
high := 1023.
low := 1.
ntry := 1.

!display "Think to a number between ",low," and ",high,
                              ": I can guess it using 10 tries at most".

!until { answer = "c" or (answer = "C")}, {
   try := ((high - low) / 2 + low).
   !display "My guess is ", try.
   !display "Is the guess (c)orrect, too (h)igh or too (l)ow?".
   answer := !accept.
   !evaluate answer
   when "C"
   when "c", {
      !display "I guessed the number using ",ntry," guesses".
   }
   when "H"
   when "h", {
      high := try.
      ntry := ntry + 1.
   }
   when "L"
   when "l", {
      low := try.
      ntry := ntry + 1.
   }
   when_other {
      !display "Answer with 'c', 'h' or 'l' please".
   }
   end_evaluate.
}.
```

A side-effect of the implementation above described is that you can write an equivalent multi-way branch using a notation that is characteristic of the COBOL EVALUATE, i.e:

```
   !evaluate (1 = 1)
   when (answer = "c" or (answer = "C")), {
      !display "I guessed the number using ",ntry," guesses".
   }
   when (answer = "h" or (answer = "H")), {
      high := try.
      ntry := ntry + 1.
   }
   when (answer = "l" or (answer = "L")), {
      low := try.
      ntry := ntry + 1.
   }
   when_other {
      !display "Answer with 'c', 'h' or 'l' please".
   }
   end_evaluate.
```

You can see how complex behaviors can be achieved using the simple mechanism object-method-args.

The code inside a block can access any variable already used outside the block, however if you use a variable in a block for the first time, it will be not available outside, i.e. that variable will be local to the block e.g.:

```
cs> a := { b := 2. ! display (b + 1). }.
cs> a exec.
3
cs> ! display b.
(null)
cs>
```

As said above the key point is that you can write your own command object in order to customize the scripts as you wish. Let's say you want to do loops using a command similar to the Java style "for", i.e. with an initialization, a condition and an increment: you can write a Java class like the following one:

```
public class MyCommand {
   public Object $for (Block init,
                       Block cond,
                       Block incr,
                       Block code) throws Exception {
      Object Return = null;
      init.exec();
      while ((Return=cond.exec()) != null &&
              Return instanceof WBoolean &&
              ((WBoolean) Return).booleanValue()) {
         Return = code.exec();
         incr.exec();
      }
      return Return;
   }
}
```

Assuming you have your class "MyCommand" available in your CLASSPATH, you can issue messages like these:

```
cs> mycmd := !newInstance "MyCommand".
cs> i := 0.
cs> mycmd for {i := 1},{i < 5},{i := i + 1},{!display i}.
1
2
3
4
cs>
```

Note that the variable *i* must be used outside any block otherwise it will be local to the block itself.

A block may have an internal name and arguments, they can be specified immediately after the open braces. The format is:

```
[name] : [arg1 [,arg2 …]].
```

These are some valid block definitions:

```
cs> a := {aName:anArg. !display "Name & arguments". }.
cs> b := {aName:. !display "Just the name". }.
cs> c := {:a1,a2. !display "Arguments only" }.
cs> d := {:. !display "Useless".}.
```

You can supply any number of argument to a block, if the argument is not supplied then the correspondent variable will contain the null value.

```
cs> blk := {:a,b. !display a,";",b.}.
cs> blk exec 1.
1;(null)
cs>
```

Blocks allows recursion, below is a script that compute the factorial of the given number.

```
fact :=
{:n.
   !if (n > 1), {
      n * (fact exec (n - 1)).
   } , {
      1.
   }.
}.

!display (fact exec 6).
```

The internal name can be used in order to cause a forced exit from the block. For example the default command implements the method *exit_block "internal-name"*. The above example can also be implemented in the following way:

```
fact :=
{all:n.
   !if (n <= 1), {
      1.
      !exit_block "all".
   }.
   n * (fact exec (n - 1)).
}.

!display (fact exec 6).
```

Note that it is not practical to use this approach in order to exit from a loop; consider the following example:

```
cs> i := 0.
cs> !until {i:=i+1. i = 3},{loop:. !display i. !exit_block "loop". !display "never
printed".}.
1
2
cs>
```

You can see that in this case the *exit_block* method interrupts the block execution but it is executed again since the exit condition is in another block. For this reason the *exit_loop* method has been implemented in the default command, e.g.:

```
cs> i := 0.
cs> !until {i:=i+1. i = 3},{!display i. !exit_loop. !display "never printed".}.
1
cs>
```

In this case the inner loop is interrupted, without the need for the block to have a name.

Outer Blocks

All the source code is implicitly contained in an "outer block". It is an ordinary block with few more features, i.e.:

- you can refer to it through the use of the special sequence "!!";
- you can use it as an object whose methods are the variables referring to a block.

So you can write:

```
cs> a := { b := 2. ! display (b + 1). }.
cs> !!a.
3
cs>
```

Note that if you try to execute !! exec, a variable whose name is "exec" will be searched and, if it exists and it refers to a block, the corresponding block will be executed.

An outer block can have both a name and arguments in the way that ordinary blocks have. When an outer block is executed by the command line, it receives an argument that is an array whose elements are the command line broken by spaces. For example let's say you have the following script named "args.joe":

```
:args.

i := -1.
!until {i := i + 1. i = (args length)},
{
   !display (args get i).
}.
```

you can issue the following command:

```
$ iscrun -joe args.joe 6 aa bb cc
args.joe
6
aa
bb
cc$
```

A script can also be executed from inside another script through the method new implemented in the default command. The script will be executed and its status (i.e. the variables) will be saved. For example let's say you have the following script named "average.joe":

```
:i_cnt,i_avg.

cnt := i_cnt doubleValue.
avg := i_avg doubleValue.

put := {:val.
   avg := avg * cnt + val.
   cnt := cnt + 1.
   avg := avg / cnt.
   !!.
}.

get := { avg. }.
```

You can use it to compute the average of a series of numbers, for example:

```
cs> avg := !new "average.joe",0,0.
cs> avg put 8.
cs> avg put 13.
cs> avg put 21.
cs> avg put 34.
cs> avg put 55.
cs> !display (avg get).
26.2
cs>
```

So a JOE script can be seen as an object from inside another JOE script. The code outer of any block is useful for initializing the object, as a Java constructor. In this object any variable containing a reference to a block will be equivalent to a public method while the other variables will be private (or rather protected, as explained later).

A JOE script can also inherit from another script through the default command method *extends*: this means the inheriting script will see all the variable from the parent script. The *extends* method has 2 arguments, i.e. the inheriting outer block and the parent outer block. For example:

```
cs> !extends !!,(!new "average.joe",0,0).
cs> !! put 8 put 13 put 21 put 34 put 55.
cs> !display avg.
26.2
cs> !display cnt.
5.0
cs>
```

So JOE has the features of a dynamic typed object oriented language using a simple model and simple implementation.

Control transfer

Even if now we can achieve any kind of computation, it could be hard to translate older script languages in it. For this reason some few features has been added, i.e. one-way transfer of control to another line of code (GO TO) and the transfer of control to another line of code with return (similar to a COBOL PERFORM).

In order to allow the transfer of the control to a specified line of code, it is necessary a way to identify a line of code: this is achieved through the use of labels: a label is simply a word followed by a dot. A label can be placed everywhere, however only the labels outside of any block can be referenced. The command object has 3 methods that manage the transfer control, i.e.:

goto	one-way control transfer; it accepts one parameter that can be either a label name or a string.
perform	control transfer with return at the original point; the return is issued when an exit is encountered; it accepts one parameter that can be either a label name or a string.
exit	causes the control to be returned where the last perform has been issued: if it is invoked outside of a perform, the procedure ends. It has no parameters.

The following example shows how the previous procedure for guessing a number can be implemented using control transfer.

```
answer := "".
high := 1023.
low := 1.
ntry := 1.

!display "Think to a number between ",low," and ",high,
                      ": I can guess it using 10 tries at most".
begin.
   try := ((high - low) / 2 + low).
   !display "My guess is ", try.
   !display "Is the guess (c)orrect, too (h)igh or too (l)ow?".
begin1.
   !perform ask.
   !if (answer = "c"), {
       !display "I guessed the number using ",ntry," guesses".
       !goto end.
   }.
   !if (answer = "h"), {
       high := try.
       ntry := ntry + 1.
       !goto begin.
   }.
   !if (answer = "l"), {
       low := try.
       ntry := ntry + 1.
       !goto begin.
   } , {
       !display "Answer with 'c', 'h' or 'l' please".
       !goto "begin1".
   }.

ask.
   answer := !accept.
   !exit.

end.
   !exit.
   !display "this is never executed".
```

JUTIL

The JUTIL utility manages JISAM files.

Usage:

```
jutil [-e=encryption_key]
      -info filename [-x]
      -load filename binary_sequential_file [-n] [-r|s] [-rs=#]
      -unload filename binary_sequential_file [-k=#]
      -loadtext filename line_sequential_file [-n] [-r|s] [-rs=#]
      -unloadtext filename line_sequential_file [-k=#]
      -loadr2 filename binary_sequential_file [-n] [-r|s]
      -shrink filename [-a]
      -check filename
      -rebuild filename [-a] [-f] [-efd=efd_dictionary]
      -getimg filename
      -makeimg filename imgstring
      -make filename efd_dictionary
      -gen [filelist] [directory]
      -convert filename directory [-d] [-s]
```

or

```
iscrun -utility jutil [-e=encryption_key]
                      -info filename [-x]
                      -load filename binary_sequential_file [-n] [-r|s] [-rs=#]
                      -unload filename binary_sequential_file [-k=#]
                      -loadtext filename line_sequential_file [-n] [-r|s] [-rs=#]
                      -unloadtext filename line_sequential_file [-k=#]
                      -loadr2 filename binary_sequential_file [-n] [-r|s]
                      -shrink filename [-a]
                      -check filename
                      -rebuild filename [-a] [-f] [-efd=efd_dictionary]
                      -getimg filename
                      -makeimg filename imgstring
                      -make filename efd_dictionary
                      -gen [filelist] [directory]
                      -convert filename directory [-d] [-s]
```

Where:

- *encryption_key* is the key to decrypt encrypted files.
- *filename* is the name of the JISAM file to which the utility refers.
- *binary_sequential_file* is the name of a binary sequential file.
- *line_sequential_file* is the name of a line sequential file.
- *imgstring* is a sequence of digits.
- *efd_dictionary* is a dictionary generated by the -efd compiler option.
- *filelist* is a text file.
- *directory* is an existing and writable directory.
- Parameters enclosed in square brackets are optional.

- The -e option should be used only when processing encrypted files. See Working on Encrypted Files for details.

JUTIL Commands:

JUTIL has a number of useful options:

Options	
-info	Displays file information. If "-x" was passed, then extended information is shown. The extended information includes the description of each key segment and the alternate collating sequence characters list.
-load	Imports data from *sequentialFile* to *jisamFile*. *JisamFile* must exist. The data read from *sequentialFile* is appended to the existing records in *JisamFile,* unless the "-n" option is used. If "-n" option is used, then *JisamFile* is emptied before loading records. The "-rs" option allows you to specify the *sequentialFile* record length. If the option is not used, then JUTIL assumes that the record length of *sequentialFile* is the same as the record length of *JisamFile*. If a unique key violation occurs, JUTIL behaves according to command line options as follows: • if "-r" was used, the *jisamFile* record is rewritten with the content of the *SequentialFile* record; • if "-s" was used, the duplicated record is skipped, and the loading process proceeds to the next record; • if neither "-r" nor "-s" options were used, the loading process aborts.
-unload	Exports data from *jisamFile* to *sequentialFile*. If *sequentialFile* exists, it is overwritten. The "-k" option allows you to specify which key must be used to read *jisamFile*. A value of 1 identifies the primary key, a value of 2 identifies the first alternate key, and so on. By default, data is read using the primary key.
-loadtext	Imports data from line *sequentialFile* to *jisamFile*. *jisamFile* must exist. The data read from *sequentialFile* is appended to the existing records in *JisamFile,* unless the "-n" option is used. If "-n" option is used, then *JisamFile* is emptied before loading records. The "-rs" option allows you to specify the *sequentialFile* record length. If the option is not used, then JUTIL assumes that the record length of *sequentialFile* is the same as the record length of *JisamFile*. JUTIL reads a new line from *sequentialFile* and then truncates the read data according to the record length. If a unique key violation occurs, JUTIL behaves according to command line options as follows: • if "-r" was used, the *jisamFile* record is rewritten with the content of the *SequentialFile* record; • if "-s" was used, the duplicated record is skipped, and the loading process proceeds to the next record; • if neither "-r" nor "-s" options were used, the loading process aborts.

-unloadtext	Exports data from *jisamFile* to line *sequentialFile*. If *sequentialFile* exists, it is overwritten. The "-k" option allows you to specify which key must be used to read *jisamFile*. A value of 1 identifies the primary key, a value of 2 identifies the first alternate key, and so on. By default, data is read using the primary key.
-loadr2	Imports data from the *sequentialFile* generated by Recover2 RM utility to the *jisamFile*. *jisamFile* must exist. The data read from *sequentialFile* is appended to the existing records in *JisamFile,* unless the "-n" option is used. If "-n" option is used, then *JisamFile* is emptied before loading records. If a unique key violation occurs, JUTIL behaves according to command line options as follows: • if "-r" was used, the *jisamFile* record is rewritten with the content of the *SequentialFile* record; • if "-s" was used, the duplicated record is skipped, and the loading process proceeds to the next record; • if neither "-r" nor "-s" options were used, the loading process aborts.
-shrink	Compresses *jisamFile* by removing deleted records. If "-a" was passed, then no user confirmation is required.
-check	Checks for file integrity.
-rebuild	Repairs a corrupted file. Before replacing the corrupted file with the repaired file, it asks for user confirmation. If "-a" was passed, then no user confirmation is required. If "-f" was passed, then duplicated records are marked as deleted and the rebuild process completes. Without "-f" the first duplicated record found interrupts the rebuild process. When a standard rebuild fails, it's worth trying again with the "-f" switch. A JIsam archive consists of two files: an index file and a data file. The rebuild process will only rebuild the index file by reading through the data file. It will not change the data file, with the exception of removing duplicate primary keys when the "-f" switch is used. It also will not build the index file if it isn't found. If the data file is corrupted, the -rebuild process will fail. If the index file is missing, Jutil can recreate it according to the information stored in an EFD dictionary. Use the "-efd" option to specify the pathname of the EFD dictionary. Be sure that the information in the EFD dictionary matches the physical characteristics of the disc file, or the rebuild process will fail. If the rebuild process fails, some temporary files may be left on disk.
-getimg	Returns a string representing file characteristics. See Image String format below for details about the string format.
-makeimg	Generates a new JIsam file according to the given image string. See Image String format below for details about the string format.
-make	Generates a new JIsam file according to the logical information and key structure described in the given EFD dictionary.
-gen	Creates a new empty JIsam file. See JIsam file generation for details.

-convert	Converts Micro Focus IDX3, IDX8 or CISAM indexed files to JIsam. See Micro Focus file conversion for details.

Working on Encrypted Files

When processing an encrypted file, the -e option must be used along with the other options to specify the encryption key. For example, the following command unloads the content of an encrypted file with the key "i5C0B0L":

```
jutil -e=i5C0B0L -unload CUSTOMERS custdata.txt
```

or

```
iscrun -utility jutil -e=i5C0B0L -unload CUSTOMERS custdata.txt
```

Note - Rebuilding (-rebuild) or shrinking (-shrink) a file with the wrong encryption key may garble the file content permanently. Pay particular attention when you use -rebuild and -shrink on encrypted files.

Image String format

The image string has the following format:

```
MaxRecSize, MinRecSize, NumKeys, [ NumSegs, Dups [ SegSize, SegOffset ] ... ] ...
```

Where:

- *MaxRecSize* is a five-digit number representing the maximum record length.
- *MinRecSize* is a five-digit number representing the minimum record length.
- *NumKeys* is a three-digit number representing the number of keys in the file.
- *NumSegs* is a two-digit number representing the number of segments in a key.
- *Dups* is a one-digit number representing the duplicate flag of a key. 0 means that duplicates are not allowed, 1 means that duplicates are allowed.
- *SegSize* is a three-digit number representing the size of a segment in a key.
- *SegOffset* is a five-digit number representing the offset of a segment in a key.

  SegSize and *SegOffset* are repeated for each segment of the key
  NumSegs, *Dups* and segments description are repeated for each key

Note - spaces in are shown to improve readability, they are not part of the format. Fields in the first pair of brackets are repeated for each key, fields in the second pair of brackets are repeated for each segment of the key.

Example - the following string applies to a file with a fixed record length of 108 bytes, a primary key composed of one segment of three bytes and an alternate key with duplicates composed of two segments, the first of two bytes in size and the second of three bytes in size:

```
00108,00108,002,01,0,003,00000,02,1,002,00003,003,00005
```

Options shortcuts

JUTIL allows you to use options thru shortcuts, you don't need to type the whole word, only the first unique bytes are tested.The following table explains which digits are testedby JUTIL when parsing options:

option	number of digits tested	digits tested
-info	2	"-i"
-load	6	"-load "
-unload	8	"-unload "
-loadtext	6	"-loadt"
-unloadtext	8	"-unloadt"
-loadr2	6	"-loadr"
-shrink	2	"-s"
-check	3	"-ch"
-rebuild	2	"-r"
-getimg	4	"-get"
-make	6	"-make "
-makeimg	6	"-makei"
-gen	4	"-gen"
-convert	3	"-co"

When the first digits of the option you type match with one of the strings in 'digits tested' column, then the corresponding option is checked and, if parameters are wrong or missing, the single option usage is shown.

In the other cases, the whole usage is shown.

Exit Status

JUTIL terminates with one of the following exit status:

Status	Meaning
0	No errors
3	File is corrupt
255	Fatal error or incorrect command line

Thin Client

JUTIL can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility jutil <arguments>
```

Server side paths must be used in the arguments.

JIsam file generation

In order to create a new empty JIsam file from scratch, use either the command command:

```
jutil –gen [filelist directory]
```

or the command:

```
iscrun -utility jutil –gen [filelist directory]
```

The –gen option of JUTIL is used to create an empty JISAM indexed file. The file structure is defined from user responses to prompts for information. These responses can be saved in a session file that can later be used in batch mode; i.e. (JUTIL –gen filelist directory). The session file has one line with the JISAM file name to be created, along with other properties of the file, such as key positions, lengths, etc. This file can have multiple lines, one for each file to be created, making this a filelist. The directory parameter specifies the target directory for the created JISAM indexed file. The directory can be left blank, or can be specified with a dot (.) to indicate the current directory. Both have the same meaning. If a directory is used, it must first exist.

To make a session file for later use, run JUTIL –gen without the filelist parameter:

For example to run JUTIL in an interactive mode to create an empty JISAM indexed file, type:

```
jutil –gen
```

The following prompts are displayed:

```
Save this session [Y]?
```

Default answers are in-between the brackets, just press ENTER to accept.

```
Enter session filename:
```

Type the session filelist name to save and press ENTER.

```
Enter JISAM filename:
```

Type the filename of the empty JISAM file to be created from the session filename and press ENTER.

```
Enter the maximum record size:
```

Type the maximum record size and press ENTER.

```
Enter the # of keys [1]:
```

Type the number of keys (Primary and Alternate) and press ENTER. The default number is 1.

```
-- Primary key --
Enter number of segments (1-16):
```

Type the number of segments for Primary key and press ENTER.

```
Enter segment size:
Enter segment offset:[0]
```

For each segment, type in the size and offset and press ENTER. The default offset value is zero. Any other value for offset should be counted as zero-based.

```
-- Alternate key 1 --
Enter number of segments (1-16):
```

Type the number of segments for Alternate key and press ENTER.

```
Duplicates allowed [N]?
```

For each Alternate key, specify whether duplicates will be allowed and press ENTER. The default value is 'N'.

```
Enter segment size:
Enter segment offset:[0]
```

For each segment, type in the size and offset and press ENTER. The default offset value is zero. Any other value for offset should be counted as zero-based.

```
JISAM file {MyFile} created.
```

This new file, MyFile can be used as input to JUTIL –gen for non-interactive file creation. For example, to create the JISAM indexed file named in MyFile with its defined properties, type:

```
jutil –gen MyFile TargetDirectory
```

or

```
iscrun -utility jutil –gen MyFile TargetDirectory
```

Micro Focus file conversion

In order to convert an existing Micro Focus file to JIsam, use either the command:

```
jutil –convert <filename> <directory> [-d] [-s]
```

or the command:

```
iscrun -utility jutil –convert <filename> <directory> [-d] [-s]
```

The –convert option of JUTIL is used to create a new JISAM indexed file with records loaded from an existing Micro Focus IDX3, IDX8 or CISAM indexed file.

-convert uses the Micro Focus "rebuild" utility on Windows and UNIX/Linux and requires *rebuild* to be in the user's PATH environment.

By running *rebuild*, JUTIL gets the input file information, then the input file is converted into a binary sequential file. An empty JISAM file is generated and records are transferred from the sequential file to the JISAM file. JUTIL runs *rebuild* using the ***-n*** and ***-o:ind,seq*** options; ensure that your copy of *rebuild* supports such options, otherwise the file conversion will not be possible.

The directory parameter specifies the target directory for the created JISAM indexed file. The directory must be specified, or can be specified with a dot (.) to indicate the current directory. The target directory must first exist.

If the -s switch is used, the Micro Focus file extension is stripped before converting the file to JIsam. This option is useful to avoid double extensions. For example, if the Micro Focus file is named "arc.dat", the corresponding JIsam file would be named "arc.dat.dat": the first ".dat" is part of the file name while the second ".dat" is the default extension used by the JIsam file handler. Use the -s switch to obtain a JIsam file named "arc.dat".

Temporary files are created in the user's temporary directory unless the TMPDIR environment variable is set. If the TMPDIR environment variable is set, then temporary files are created in the directory specified by the TMPDIR environment variable. Temporary files are not removed by default when the file conversion is completed. Use the -d switch to have these files removed.

A session file with the description of the indexed file is left on disc at the end of the process. You can use this file to create a JISAM file with the same structure with JUTIL –gen, if you need. See JIsam file generation for details.

STREAM2WRK

The STREAM2WRK utility opens JSON, XML, WSDL and XSD files and generates the corresponding COBOL record description to be used by runtime classes like XMLStream, JSONStream and HTTPClient or in an XD record definition.

Usage 1 (JSON):

```
stream2wrk json jsonFile [-o outputfile] [-p prefix] [-d] [-r rootname] [-anyescape]
```

Where:

- *jsonFile* is the name of the JSON file to parse. It can be either a disk file or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In those cases, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.
- *outputFile* is the name of the file that will contain the record definition corresponding to *jsonFile*. If omitted, a file named *jsonFile*.wrk is created.
- *prefix* defines a string to be put in front of every data name in the record definition.
  - When set to "`0`" or omitted, data-names are generated with no prefix.
  - When set to "`1`", the prefix will be the name of the JSON file.
  - Any other value represents the prefix to be used, without any conversion.
- *-d* activates or deactivates names ambiguity check
  - When -d is omitted, field names are generated without control.
  - When -d is used, field names are adapted if necessary in order to avoid ambiguous identifiers.
- When processing a JSON without root element or with an array as root element, Stream2Wrk generates a 01 level named "json2wrk". This name can be changed via -r command line option.

- *-anyescape* allows you to process JSON streams that include backslashes among item values. Use this option if you encounter the error *Internal error:org.xml.sax.SAXException: Unxpected character*.

Usage 2 (WSDL):

```
stream2wrk wsdl wsdlFile [-o outputfile] [v1.1]
```

Where:

- *wsdlFile* is the name of the WSDL file to parse. It can be either a disk file or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In those cases, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.
- *outputFile* is the name of the file that will contain the record definition corresponding to *wsdlFile*. Multiple copy files are generated, one for each method described in the WSDL. The copy file is named by appending the method name to *outputfile*.
  If outputfile is omitted, then the generated code is printed on the console.
- *-v.1.1* should be used to process WSDL files of version 1.1. By default the utility expects WSDL files of version 2.0.

Usage 3 (XML):

```
stream2wrk xml xmlFile [-o outputfile] [-p prefix] [-d] [-e] [-se enumeration-suffix]
[-sa attribute-suffix] [-c] [-sc count-suffix] [-scp capacity-suffix] [-sd
data-suffix] [-iu] [-l[=len]] [-nc]
```

Where:

- *xmlFile* is the name of the XML file to parse. It can be either a disk file or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In those cases, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.
  If the XML file includes a XSD, then the utility parses the XSD and the result is more accurate. If no XSD is available, then the utility guesses the fields characteristics according to the XML content.
- *outputFile* is the name of the file that will contain the record definition corresponding to *xmlFile*. If omitted, a file named *xmlFile*.wrk is created.
- *prefix* defines a string to be put in front of every data name in the record definition.
  - When set to "0" or omitted, data-names are generated with no prefix.
  - When set to "1", the prefix will be the name of the XML file.
  - Any other value represents the prefix to be used, without any conversion.
- *-d* activates or deactivates names ambiguity check
  - When -d is omitted, field names are generated without control.
  - When -d is used, field names are adapted if necessary in order to avoid ambiguous identifiers.
- *-e* generates 88 level representing 'enumeration' tags.
- *-se* specifies an alternate suffix for the enumeration data-items.
- *-sa* specifies an alternate suffix for the 'attribute' data-items.
- *-c* generate 'count' data-items.
- *-sc* specifies an alternate suffix for the 'count' data-items.
- -scp specifies an alternate suffix for the 'capacity' data-items when OCCURS DYNAMIC is used.
- *-sd* specifies an alternate suffix for the 'data' data-items.

- *-iu* doesn't generate OCCURS when 'maxOccurs=unbounded' is found in the XSD.
- *-l* generates alphanumeric data-items with fixed size instead of using 'pic x any length'. The size is equal to the value of the 'maxLength' attribute in the XSD, if present, otherwise it is equal to *len*; if *len* was not specified, then 'len=80' is assumed.
- *-nc* doesn’t generate comment lines.

Usage 4 (XSD):

```
stream2wrk xsd xsdFile [-o outputfile] [-p prefix] [-d] [-e] [-se enumeration-suffix]
[-sa attribute-suffix] [-c] [-sc count-suffix] [-scp capacity-suffix] [-sd
data-suffix] [-iu] [-l[=len]] [-nc]
```

Where:

- *xsdFile* is the name of the XSD file to parse. It can be either a disk file or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In those cases, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.
- *outputFile* is the name of the file that will contain the record definition corresponding to *xsdFile*. If omitted, a file named *xsdFile*.wrk is created.
- *prefix* defines a string to be put in front of every data name in the record definition.
  - When set to "0" or omitted, data-names are generated with no prefix.
  - When set to "1", the prefix will be the name of the XSD file.
  - Any other value represents the prefix to be used, without any conversion.
- *-d* activates or deactivates names ambiguity check
  - When -d is omitted, field names are generated without control.
  - When -d is used, field names are adapted if necessary in order to avoid ambiguous identifiers.
- *-e* generates 88 level representing 'enumeration' tags.
- *-se* specifies an alternate suffix for the enumeration data-items.
- *-sa* specifies an alternate suffix for the 'attribute' data-items.
- *-c* generate 'count' data-items.
- *-sc* specifies an alternate suffix for the 'count' data-items.
- *-scp* specifies an alternate suffix for the 'capacity' data-items when OCCURS DYNAMIC is used.
- *-sd* specifies an alternate suffix for the 'data' data-items.
- *-iu* doesn't generate OCCURS when 'maxOccurs=unbounded' is found in the XSD.
- *-l* generates alphanumeric data-items with fixed size instead of using 'pic x any length'. The size is equal to the value of the 'maxLength' attribute in the XSD, if present, otherwise it is equal to *len*; if *len* was not specified, then 'len=80' is assumed.
- *-nc* doesn’t generate comment lines.

Thin Client

STREAM2WRK can’t be launched directly by the isCOBOL Client.

WSDL2WRK

This utility is deprecated and supported only for backward compatibility. STREAM2WRK should be used instead.

The WSDL2WRK utility takes a WSDL file and generates a COBOL description of the 'SOAP Envelopes' used by the service. Two envelopes are generated for each service: an envelope for date request and an envelope for data response.

Usage:

```
iscrun -utility wsdl2wrk [-v1.1] [-o outputFile] [-amn] wsdlFile
```

Where:

- *-v.1.1* should be used to process WSDL files of version 1.1. By default the utility expects WSDL files of version 2.0
- *outputFile* is the name of the file that will contain the record definition corresponding to *wsdlFile*.
- *-amn* instructs the utility to append the method name rather than a progressive number to each generated copy file.
- *wsdlFile* is the name of the WSDL file to parse. It can be either a disk file name or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In such case, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.

If there is more than one schema, then more copyfiles are created with the following criteria:

- if there is no suffix on the *outputFile* name, the first is created with the same name as *outputFile* and the others with the name *outputFile<n>*, where *n* is a progressive number.
- if there is a suffix on the *outputFile* name, the first is created with the same name as *outputFile* and the others with the name *outputFile<n>.<suffix>*, where *outputFile* is the output file name without extension, *n* is a progressive number and *suffix* is the output file name extension.

Thin Client

WSDL2WRK can't be launched directly by the isCOBOL Client.

XML2WRK

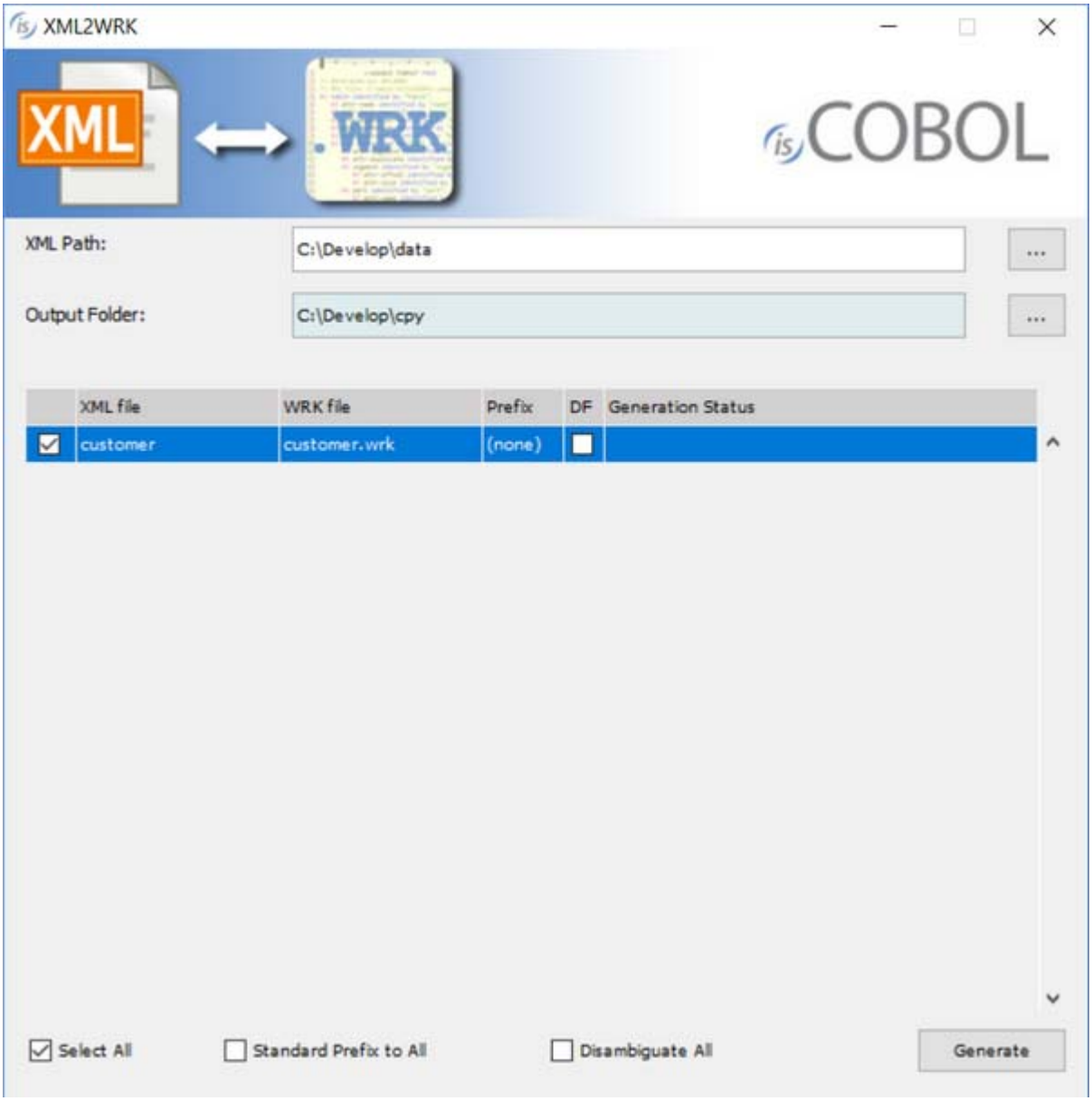
This utility is deprecated and supported only for backward compatibility. **STREAM2WRK** should be used instead.

The XML2WRK utility opens an XML file and creates the corresponding record definition to be used with the XMLStream Class (com.iscobol.rts.XMLStream) object.

Usage 1:

```
iscrun -utility xml2wrk
```

If the utility is launched without parameters, a graphical wizard procedure will start.



Usage 2:

```
iscrun -utility xml2wrk xmlFile [outputFile] [prefix] [disambiguate_flag]
```

Where:

- *xmlFile* is the name of the XML file to parse. It can be either a disk file or a URL. Note that the utility is not able to read a file over the HTTPS protocol. In such case, the file should be downloaded to disc using third party utilities (e.g. a web browser) and then processed as a disc file.
- *outputFile* is the name of the file that will contain the record definition corresponding to *xmlFile*. If omitted, a file named *xmlFile*.wrk is created.
- *prefix* defines a string to be put in front of every data name in the record definition.

    - When set to "0" or omitted, data-names are generated with no prefix.
    - When set to "1", the prefix will be the name of the XML file.
    - Any other value represents the prefix to be used, without any conversion.
- *disambiguate_flag* activates or deactivates names ambiguity check
    - When set to "0" or omitted, field names are generated without control
    - When set to "1", field names are adapted if necessary in order to avoid ambiguous identifiers

XML2WRK uses the following criteria while parsing the XML file:

- Only the first occurrence of each element is parsed to retrieve child items.
- If an element appears more than once in the XML file, then it's generated as an OCCURS item in the COBOL record definition, otherwise it is generated as standard item.
- If the element contains text or attributes, data-items are generated in the COBOL record definition to store text and attribute values, otherwise the element is generated as a container item without picture.
- Every data-item is generated as PIC X ANY LENGTH into the COBOL record definition.

Consider the following sample xml:

```
<content>
 <parent>
   <child>
     <item>text</item>
   </child>
   <child>
     <lost></lost>
   </child>
 </parent>
 <parent>
   <child></child>
 </parent>
</content>
```

The underlined text highlights elements that are processed by XML2WRK according to the above rules.

- <parent> and <child> will be generated as OCCURS items because they appear twice
- <item> will not be generated as OCCURS item because it appears once
- <lost> will not be generated because it's included into an item that is not parsed by XML2WRK
- a data-item will be generated for <item> because it contains text

The resulting record definition is:

```
01 content identified by "content".
   03 parent identified by "parent" occurs dynamic capacity parent-count.
      05 child identified by "child" occurs dynamic capacity child-count.
         07 item identified by "item".
            09 item-data pic x any length.
```

Thin Client

XML2WRK can be used in thin client environment as well. Use this command to start it:

```
iscclient -hostname <server-ip> -port <server-port> -utility xml2wrk <arguments>
```

Server side paths must be provided in the arguments.

Chapter 5

Advanced Features

Introduction

isCOBOL Code Coverage and isCOBOL Unit Test are enterprise level features that enable developers to write robust test suites and check their effectiveness, allowing the production of a more stable code base in applications.

isCOBOL Code Coverage

Introduction

Test coverage is a measure used to describe the degree to which the source code of a program is executed when a particular test suite runs. A program with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.

While an application is running with code coverage, code execution is logged for each program and sub-program. The results are rendered in HTML format at the end of the runtime session.

Covered and missed blocks

A block is a portion of code that, when executed, is executed consecutively and linearly. This means you can assume that if one part of the block executes, then the rest of the block has been executed as well.

Code that is not executed linearly includes more than one block of code. It is the case, for example, of IF THEN ELSE statements, where the individual clauses in the statement are not executed in a single pass.

Blocks of code that are executed are considered covered, while blocks of code that are not executed are considered missed.

Restrictions

The isCOBOL code coverage implementation captures only the activity of COBOL classes, so classes that implement the com.iscobol.rts.IscobolClass interface. Other classes are not considered.

Running an application with isCOBOL Code Coverage from the command-line

Before running the application with isCOBOL Code Coverage, you should consider configuring the Coverage engine by setting one or more of the properties described in isCOBOL Code Coverage Configuration. For example, you can include only some of the programs in the test coverage by setting iscobol.coverage.analysis.includes or to exclude some of the programs by setting iscobol.coverage.analysis.excludes.

You may also specify the location of COBOL source files by setting iscobol.coverage.sourcefiles, as these files are searched in the current directory by default, but the current directory doesn't usually match with any of the directories where source files are stored.

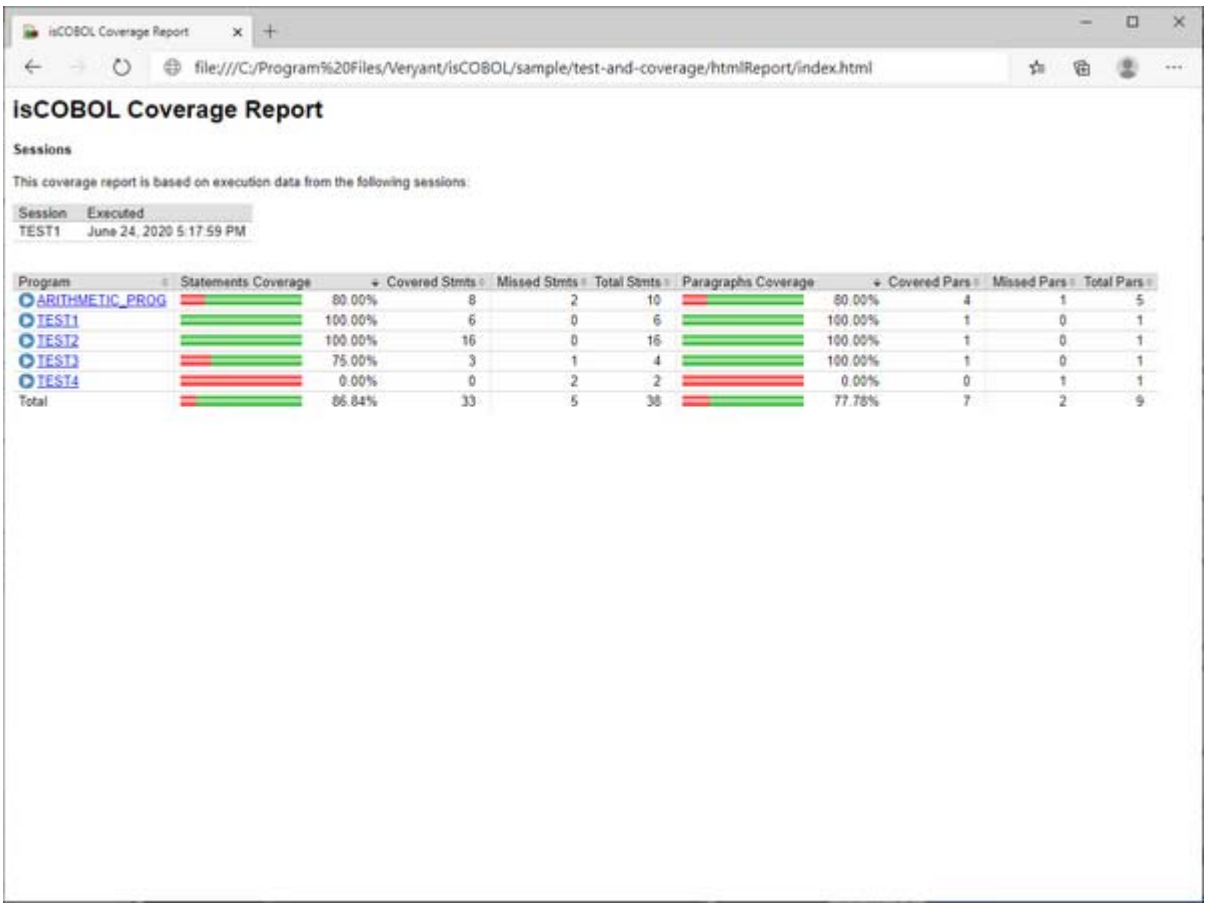
You can also create a folder to host the HTML report generated by the Coverage engine, and set iscobol.coverage.html to point to this folder. By default, the Coverage engine looks for a folder named "htmlReport" in the current directory. If the folder doesn't exist, it is automatically created.

In order to run an application with isCOBOL Code Coverage, add the -coverage option to the runtime command line, e.g.

```
iscrun -coverage -c myApp.cfg ProgramName
```

When the runtime session terminates, the folder pointed to by iscobol.coverage.html will include several files. Open "index.html" with your favorite web browser to see the isCOBOL Code Coverage report.

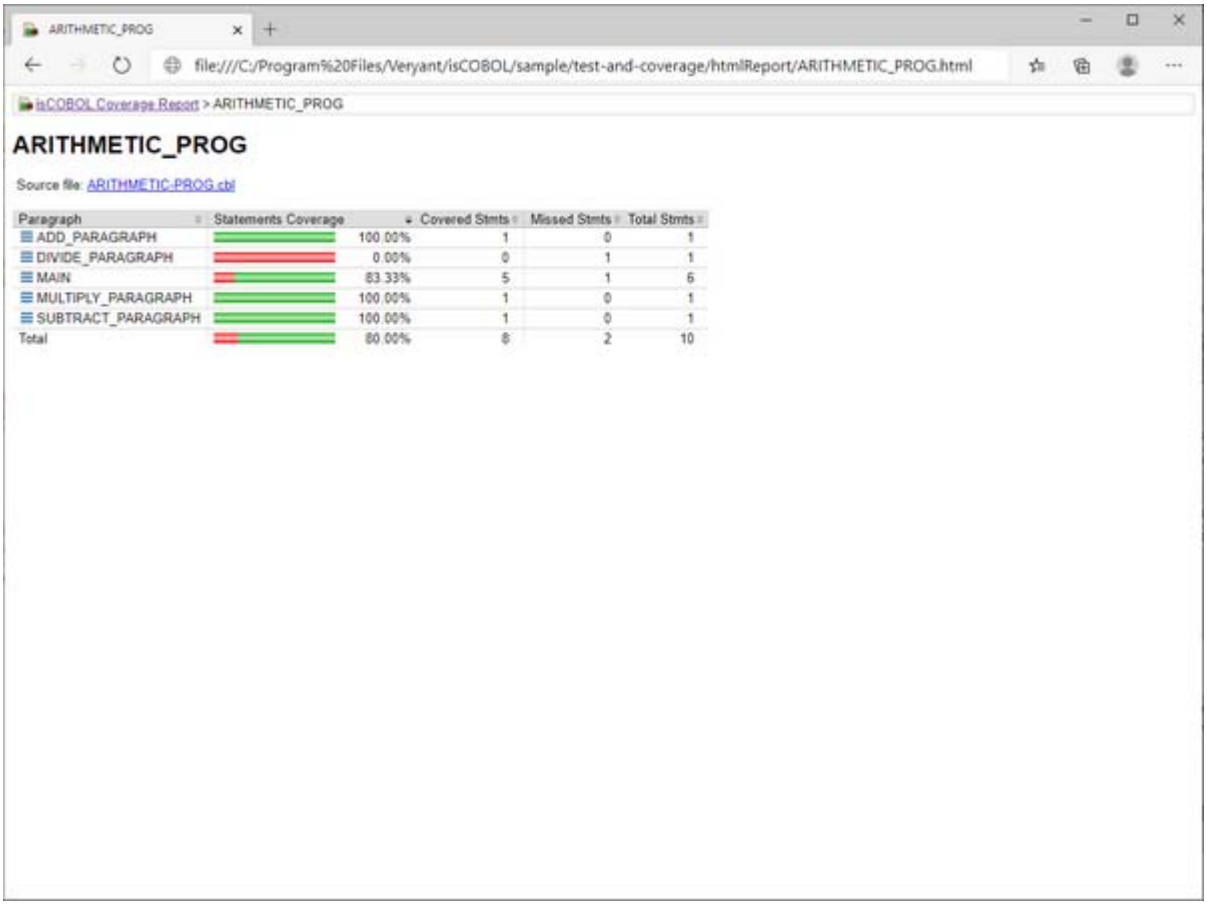
The home page of the report will look like this:



In the home page you see the list of involved COBOL classes. For each class, the following information is provided:

Column Name	Column Content
Program	Class name stripped of the ".class" extension.
Statements Coverage	Graphical representation of the amount of missed statements (red) and the amount of covered statements (green).
Covered Stmts	Count of covered statements.
Missed Stmts	Count of missed statements.
Total Stmts	Total number of statements.
Paragraphs Coverage	Graphical representation of the amount of missed paragraphs (red) and the amount of covered paragraphs (green).
Covered Pars	Count of covered paragraphs.
Missed Pars	Count of missed paragraphs.
Total Pars	Total number of paragraphs.

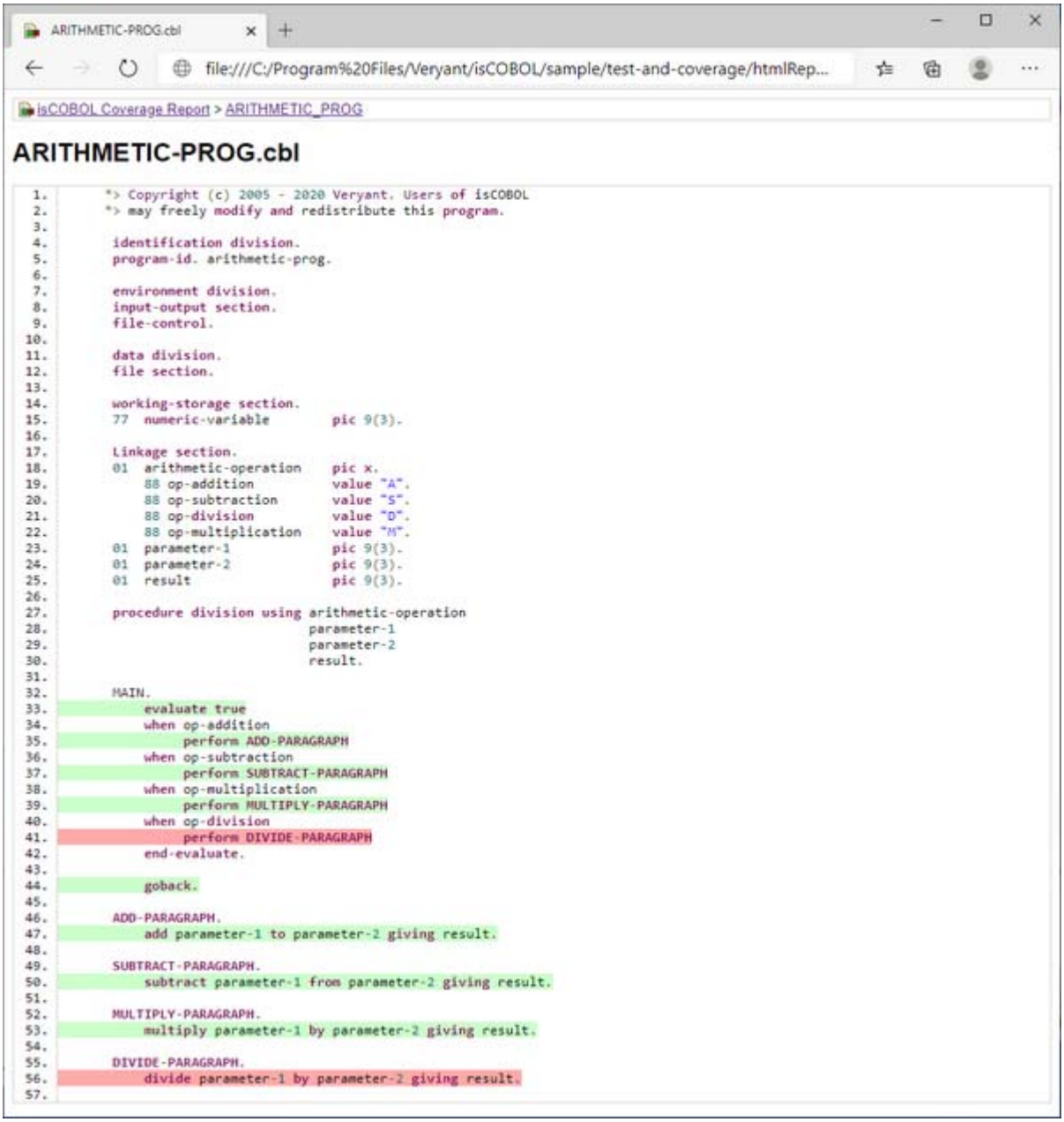
Clicking on the COBOL class name in the Program column brings you to a detailed analysis of the class activity:



In this page, the following information is provided:

Column Name	Column Content
Paragraph	Name of the paragraph. It is prefixed by the method name if the interested class is a CLASS-ID program.
Statements Coverage	Graphical representation of the amount of missed statements (red) and the amount of covered statements (green).
Covered Stmts	Count of covered statements.
Missed Stmts	Count of missed statements.
Total Stmts	Total number of statements.

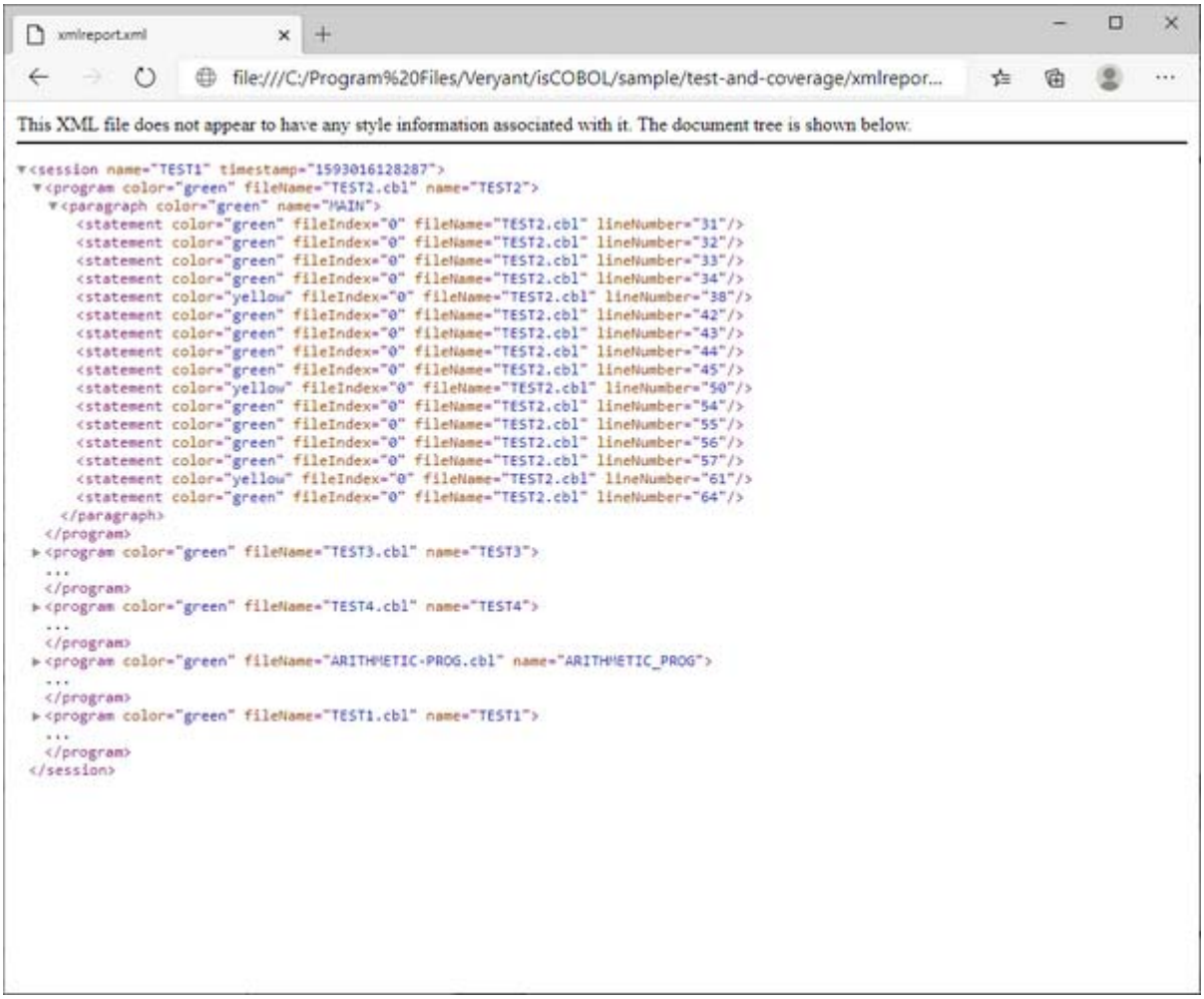
Clicking on the source file name above the table brings you to a detailed analysis of the source code, where you can see covered code with a green background and missed code with a red background:



It's also possible to obtain a coverage report in XML format.

Set iscobol.coverage.xml to the name of the XML file that you want to obtain.

The XML file content looks like this:



The main element is *session* and it includes one or more *program* element.

Every *program* element includes one or more *paragraph* elements.

Every *paragraph* element includes one or more *statement* element.

For each element, the *color* attribute specifies if the element was covered ('green') or not ('red'). The value 'yellow' indicates a statement that has multiple branches and that not all the branches in code have been reached (e.g. an IF statement).

If you set only one between iscobol.coverage.xml and iscobol.coverage.html, then only the report related to the property that you set is generated.

If you set both properties, then both reports are generated.

If you don't set any of these properties, then an HTML report is generated in the "./htmlReport" directory.

XML reports of multiple runtime sessions can be merged together by setting the iscobol.coverage.append configuration property.

The javaagent option

The javaagent option allows you to customize the Code Coverage behavior where the -coverage option is not available, like for example in application server environments or in WEB.

The command

```
iscrun -coverage ProgramName
```

is equivalent to:

```
iscrun -J-javaagent:/path/to/isprofiler.jar=coverage;html=htmlReport ProgramName
```

Note - isprofiler.jar is located in the lib folder of the isCOBOL SDK.

The javaagent option allows you to specify some options to customize the Code Coverage behavior and the resulting report. The syntax is:

```
-J-javaagent:/path/to/
isprofiler.jar=coverage;[option1=value1;option2=value2;...;optionN=valueN]
```

Where the available options are:

Option	Value
append	Pathname of a report file in XML format to be appended to the existing XML report. This option can be specified multiple times.
classfiles	Location of the class files.
excludes	List of programs that must not be analyzed. Multiple values must be separated by comma.
html	Pathname of a folder that will host a report in HTML format.
includes	List of programs that must be analyzed. Multiple values must be separated by comma. By default, all programs are analyzed.
sessionname	Name of the coverage session.
sourcefiles	Location of the source files.
xml	Pathname of a report file in XML format.

Using Code Coverage and Profiler together

The isprofiler.jar library implements both the Code Coverage and the Profiler, however it's not possible to use these two feature together.

On the iscrun command line, if you specify both the -coverage option and the -profile option, e.g.

```
iscrun -profile -coverage ProgramName
```

an error is shown and the runtime doesn't start.

Using the javaagent option as follows:

```
-J-javaagent:/path/to/
isprofiler.jar=coverage;[option1=value1;option2=value2;...;optionN=valueN];profiler;[o
ption1=value1;option2=value2;...;optionN=valueN]
```

A warning is shown and only the first feature (coverage, in this case) is activated.

The C$COVERAGE library routine

Another way to customize the Code Coverage behavior and the report files is by calling the C$COVERAGE library routine. The routine is even more powerful than the javaagent option because it allows you to choose when the data gathered by the Code Coverage should be flushed to disc (see CCOV-FLUSH).

Running an application with isCOBOL Code Coverage from the IDE

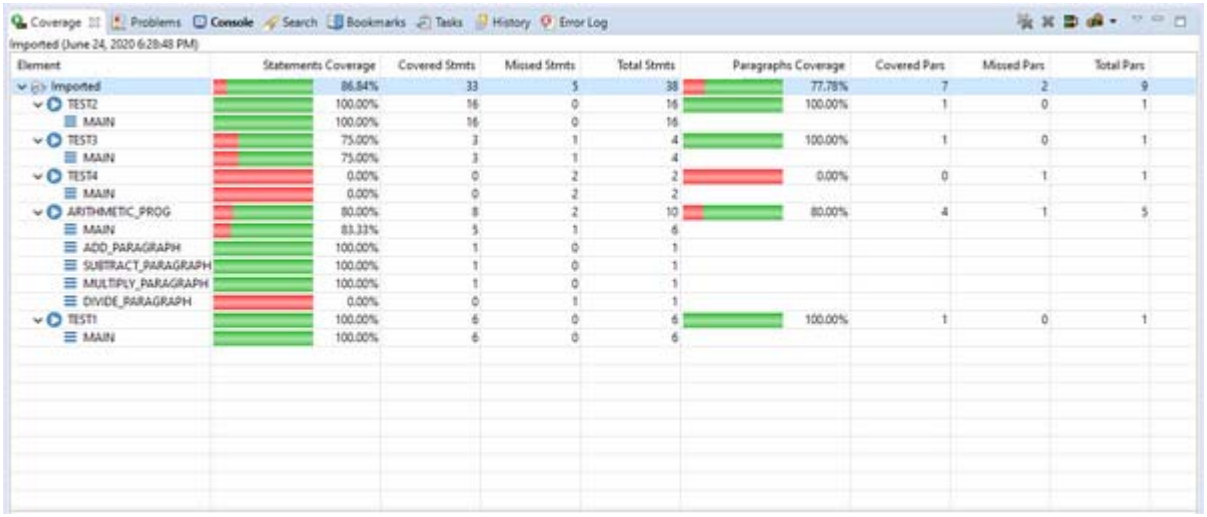
Test coverage is integrated in the isCOBOL IDE.

You can easily perform a test coverage as follows:

1. highlight the main source file name in the isCOBOL Explorer or click on the editor window for the main program file,
2. click on the *Run* menu and choose *COBOL Coverage As -> isCOBOL Application* or click on the corresponding button in the toolbar:

The above operations trigger the program execution.

When the runtime session terminates, the Coverage view shows the coverage report.



isCOBOL IDE automatically sets the isCOBOL Code Coverage configuration as follows:

- iscobol.coverage.classfiles is set to the *output* folder of the project,
- iscobol.coverage.sessionname is set to the name of the project,
- iscobol.coverage.sourcefiles is set to the list of folders in the *-sp* Compiler option plus the *source* folder of the project.

Using isCOBOL Code Coverage in application server environments

Thin client

In a thin client environment it is possible to analyze the application server (isCOBOL Server) activity by starting the server process with the same *-javaagent* option used for the runtime. E.g.:

```
iscserver -J-javaagent:/path/to/isprofiler.jar=coverage;[options]
```

The Coverage output is shown when the whole application server is terminated or when the CCOV-FLUSH function of C$COVERAGE is called, and it includes the analysis of all clients activities mixed together, therefore, if you need to analyze some programs in a thin client environment, you should use a dedicated application server with only one client connected.

Tomcat and other application servers

In an application server environment like Tomcat it is possible to profile the programs' activity by starting the server process with the same *-javaagent* option used for the runtime. Add the following Java option to the startup options of your application server.:

```
-javaagent:/path/to/isprofiler.jar=coverage;[options]
```

The isprofiler.jar agent implements both the Code Coverage and the Profiler tools. See Using Code Coverage and Profiler together for information on how to use them together.

The following libraries should be shared among all the webapps:

- asm-9.2.jar
- asm-tree-9.2.jar
- isprofiler.jar
- jacoco-core-0.8.7.jar

You can either copy them to the Tomcat's lib directory or you can add them to the CLASSPATH setting.

When you use coverage features under a servlet container, no report is generated at the closing of the JVM. The generation of the reports is delegated to the C$COVERAGE library routine:

- Call the CCOV-SET function to specify the name of the report file.
- Call the CCOV-FLUSH function to stop the coverage activity and generate the report file.

It's good practice to call these functions in the main program of the servlet or in a ServletContextListener class, so the whole session is analyzed.

It's strongly suggested to have only one session of the webapp running with coverage. If there are multiple sessions of the same webapp running with coverage, the report will not be accurate as it collects information from multiple sessions.

Example of ServletContextListener written in object oriented COBOL that allows you to coverage the webapp activity:

```
       identification division.
       class-id. awebxcontextlistener as
                   "AwebxContextListener"
                   implements servletcontextlistener.
       environment division.
       configuration section.
       repository.
          class servletcontextlistener as
             "javax.servlet.ServletContextListener"
          class servletcontextevent as
             "javax.servlet.ServletContextEvent"
             .
       identification division.

       factory.

       working-storage section.
       copy "iscobol.def".

       end factory.

       identification division.

       object.

       procedure division.

       identification division.
       method-id. contextInitialized as "contextInitialized".
       linkage section.
       01 evt object reference servletcontextevent.

       procedure division using evt.
       main.
            call "c$coverage" using ccov-set "html"
                                    "/tmp/coverage_reports".
            goback.

       end method.

       identification division.
       method-id. contextDestroyed as "contextDestroyed".
       linkage section.
       01 evt object reference servletcontextevent.

       procedure division using evt.
       main.
           call "c$coverage" using ccov-flush.
           goback.

       end method.

       end object.

       end class.
```

Troubleshooting

The following exception may occur while running programs under Code Coverage:

```
org.objectweb.asm.MethodTooLargeException: Method too large: PROGRAM.PARAGRAPH ()I
```

The reason is that the Coverage feature adds on the fly some instructions to the methods in your class, it's like if you added some COBOL statements to the program's paragraphs. If a method grows too much, it can lead to the above error. In order to get rid of it, recompile your program with the -sns=Statements option.

isCOBOL Unit Test

Introduction

Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.

Running an Unit Test from the command-line

In order to create an Unit Test, you need to list which programs must be included in the test. The list is provided as a text file pointed by the configuration property iscobol.unit_test.list_file. For example, the below settings specifies that the list of programs is stored in the file *list.txt* placed the runtime working directory:

```
iscobol.unit_test.list_file=./list.txt
```

The list file must specify each program on a separate line. For example, the below file content causes TEST1, TEST2, TEST3 and TEST4 to be included in the Unit Test:

```
TEST1
TEST2
TEST3
TEST4
```

It is also possible to specify more than one list file, separating them with the system path separator (semicolon on Windows, colon on Linux/Unix) or by the \n character sequence, e.g.

```
iscobol.unit_test.list_file=./list.txt\n./list2.txt
```

You may also consider to create a folder to host the HTML report generated by the Unit Test, and set iscobol.unit_test.html to point to this folder. By default, the Unit Test engine looks for a folder named "htmlReport" in the current directory. If the folder doesn't exist, it is automatically created.

In order to run the Unit Test, start the runtime with the -iut option, e.g.

```
iscrun -c iut.cfg -iut
```

In the above sample command, iut.cfg is a configuration properties file that must include at least the iscobol.unit_test.list_file setting.

The isCOBOL Runtime will execute all the programs in the list files, one by one. In the end the folder pointed by iscobol.unit_test.html will include a file whose name is the name of the list file plus the html extension (e.g. *list.txt.html*). Open this with your favorite web browser to see the Unit Test report.

Using Assertions

An assertion is a predicate connected to a point in the program, that always should evaluate to true at that point in code execution. Assertions can help a programmer read the code, help a compiler compile it, or help the program detect its own defects.

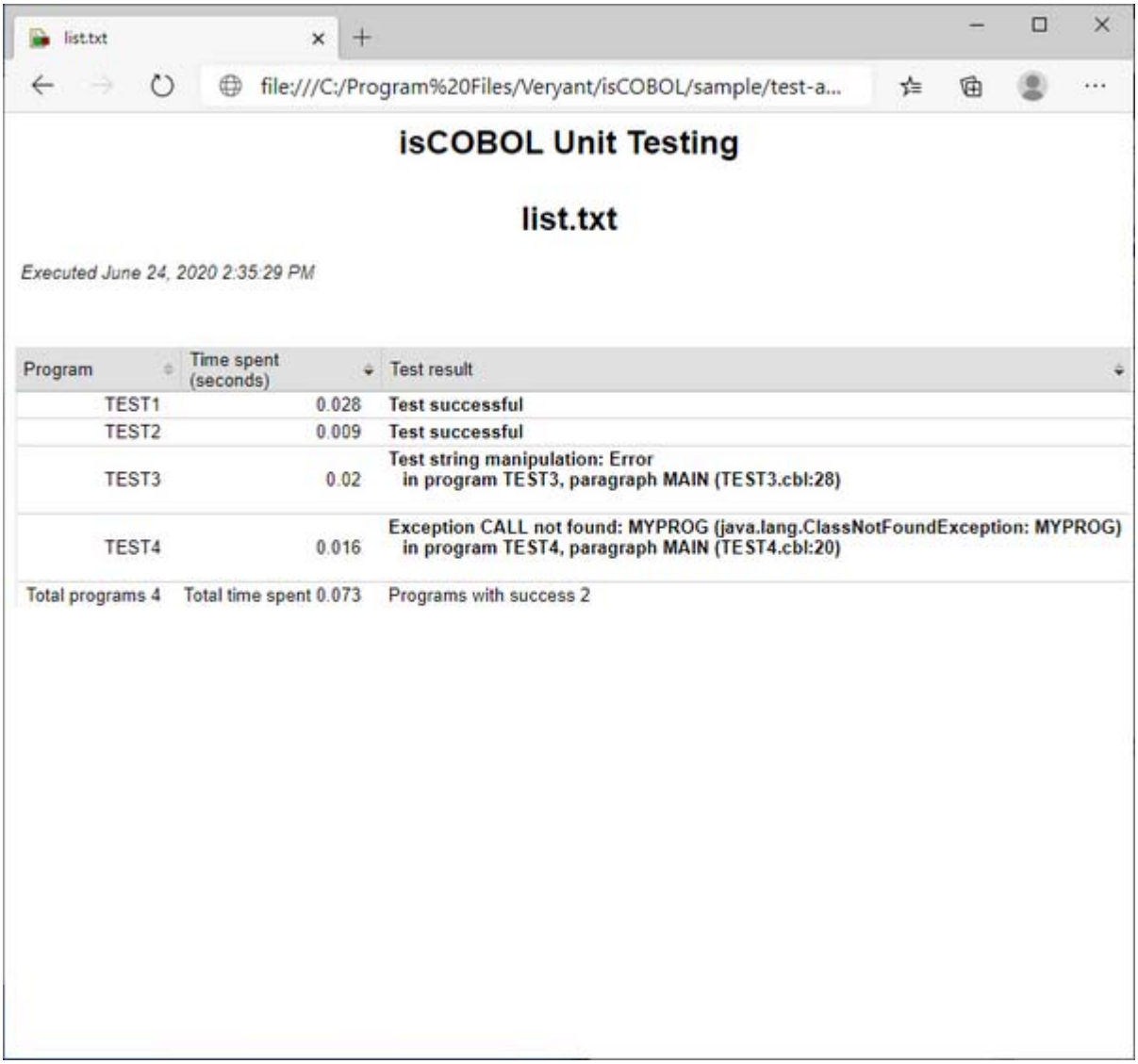
You can add assertion to the source code of your program using the ASSERT statement, for example:

```
    assert string1 = "my string"
           otherwise "Test string manipulation: Error"
```

In order to evaluate assertions during the Unit test, add the -ea Java option to the command line, e.g.

```
iscrun -c iut.cfg -iut -J-ea
```

Considering for example that all tests were successful except for TEST3 that failed due to a failed assertion and TEST4 that failed due to a missing subprogram, the HTML report will look like this:



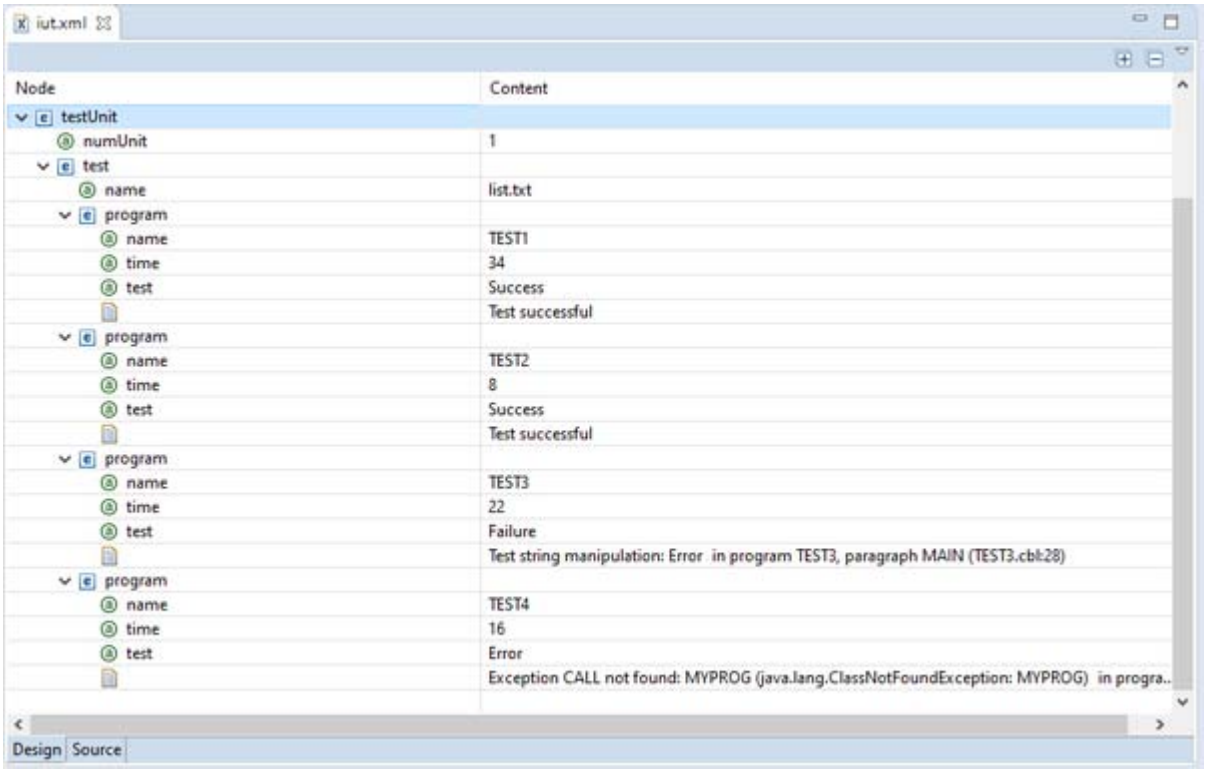
In the page, the following information is provided:

Column Name	Column Content
Program	Name of the COBOL program as it appears in the file pointed by iscobol.unit_test.list_file.
Time spent (seconds)	Number of seconds that the program spent to complete.
Test result	"Test successful" if the program completed without errors or the exception stack of the error otherwise.

It's also possible to obtain a report in XML format.

Set iscobol.unit_test.xml to the name of the XML file that you want to obtain.

The XML file contains only the list of failed tests, so the XML for the Unit Test described above would look like this:



If you set only one between iscobol.unit_test.xml and iscobol.unit_test.html, then only the report related to the property that you set is generated.

If you set both properties, then both reports are generated.

If you don't set any of these properties, then an HTML report is generated in the "./htmlReport" directory.

Unit Test console output and exit codes

When iscrun is launched with the -iut option, it prints a summary of the outcome upon exit.

A successful test will generate the following message:

```
All <testsCount> tests "ok", for details see <reportName>
```

In this case, the exit code of the iscrun process is 0.

A test with errors will generate the following message:

```
<failuresCount> of <testsCount> tests "failed", for details see <reportName>
```

In this case the exit code of the iscrun process is *failuresCount*.

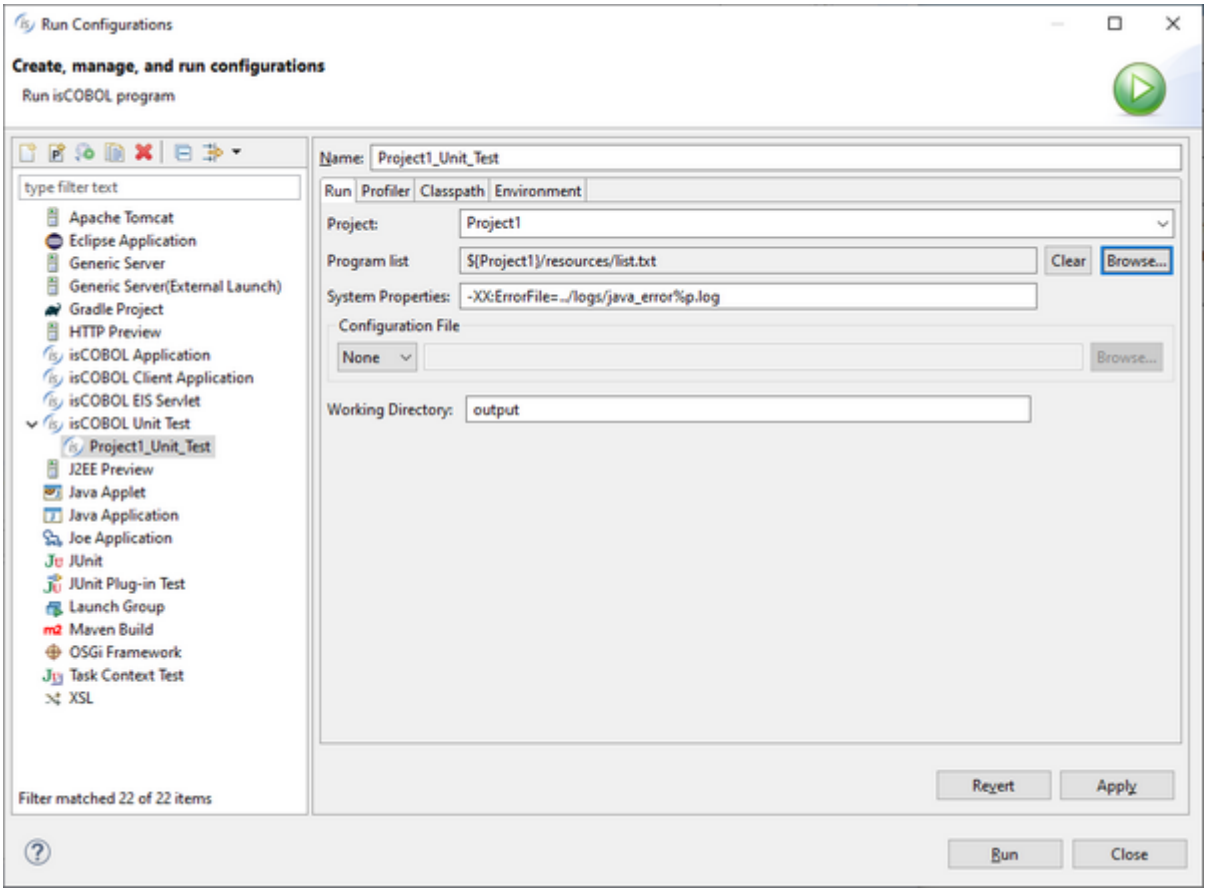
Running an Unit Test from the IDE

The isCOBOL Unit Test is integrated in the isCOBOL IDE.

In order to create a Unit Test in the IDE, a dedicated Run Configuration must be created.

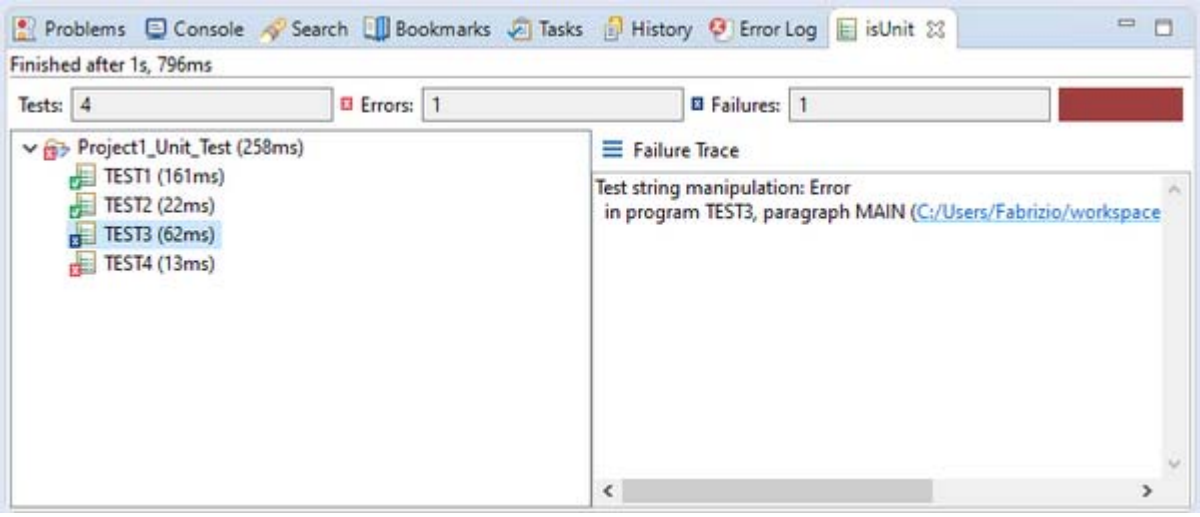
1. click on *Run* in the menu bar and choose *Run Configurations...*,
2. Right click on *isCOBOL Unit Test* in the list on the left and choose *New Configuration*,
3. Fill the fields as follows:

Field	Value
Name	The name that you wish to assign to this Run Configuration.
Project	The project where the programs to test are included. Choose it from the list.
Program list	Click on the *Browse* button after the field and use the pop-up dialog to select the list of programs to include. You can select the list file either from the current workspace or from the file system. Ensure that the list file includes only main programs, so programs that don't use Linkage Section items.



4. Click on the *Apply* button, then on the *Run* button

The Unit Test will run and the results will be shown in the isUnit view.



A green panel in the top right corner of this view means that all tests were successful. A red panel instead means that at least one of the tests failed. Clicking on the program name populates the Failure Trace field with the exception stack of the error that caused the failure of the test. Clicking on the hyperlinks in the stack will bring you to the problematic line of code in the COBOL source.

From this moment you can run or debug the test by choosing the Run Configuration name from the *Run History* menu and the *Debug History* menu. It's also possible to run the Unit Test along with a Coverage test, as explained later in this document.

Using isCOBOL Code Coverage and isCOBOL Unit Test together

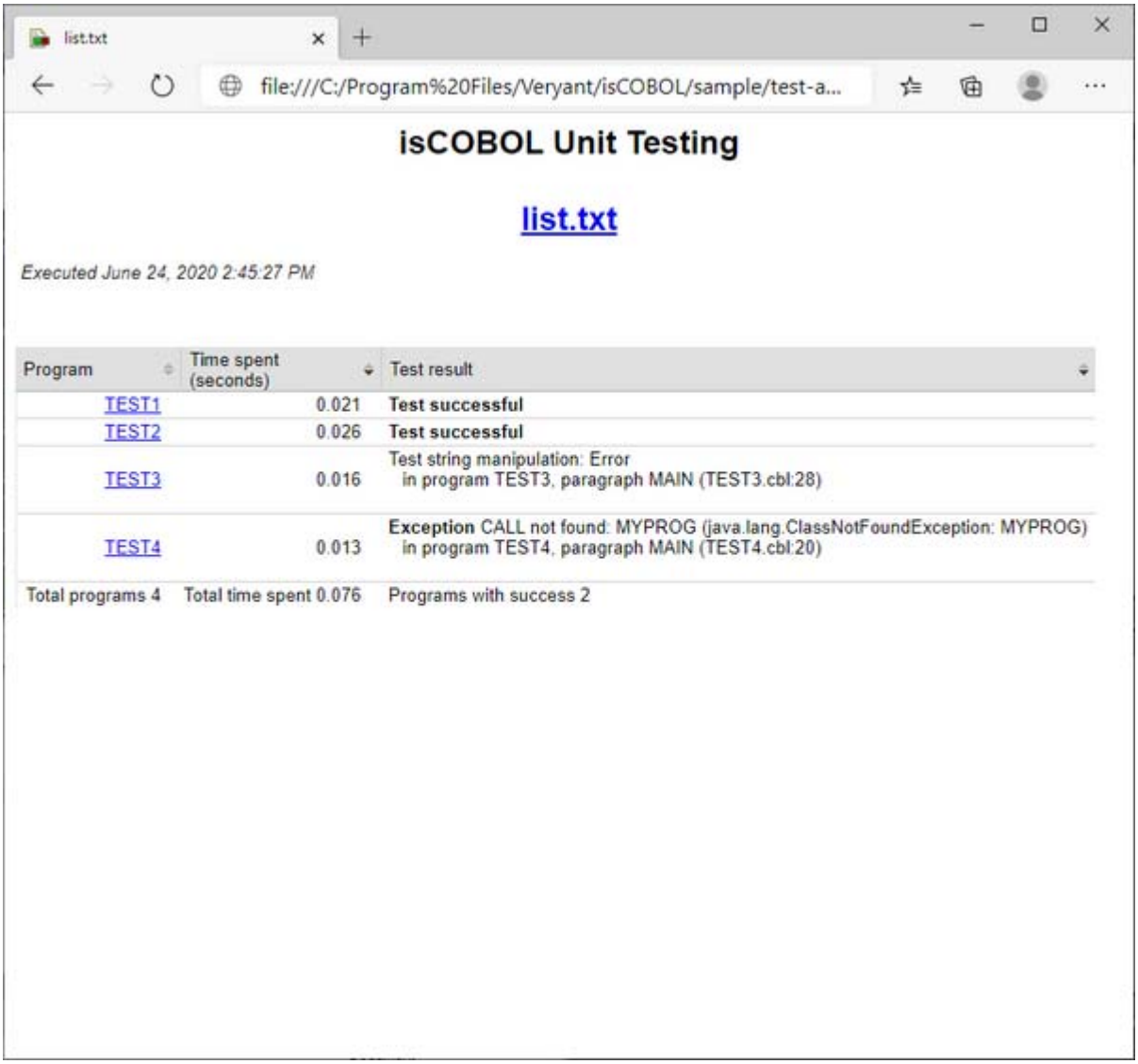
isCOBOL Code Coverage and isCOBOL Unit Test can be used together by activating both runtime options on the runtime command line or by invoking the dedicated Run Configuration from the *isCOBOL Coverage History* in the IDE.

In order to use both features on the command line, use a command like this:

```
iscrun -c iut.cfg -iut -J-ea -coverage
```

When the two features are used together all the HTML reports are stored in the folder specified by iscobol.coverage.html and iscobol.unit_test.html is ignored.

The file generated by the Unit Test engine becomes the main document and clicking on a program name in that document you can reach the coverage report of that program:



In order to use both features in the IDE, follow the steps described in Running an Unit Test from the IDE to create the necessary Run Configuration, then

1. click on *Run* in the menu bar,
2. choose *isCOBOL Coverage History*,
3. select the Run Configuration previously created.

The Coverage view and the isUnit view will show the report of the test.

Chapter 6

Files Management

Managing files is an important task for most COBOL applications. This section discusses the implementation of the three types of files: sequential, relative, and indexed.

Sequential Files

isCOBOL treats sequential files in one of two ways:

Binary sequential: designed to contain non-ASCII information and are easy to move to foreign systems. A binary sequential file consists of either fixed-length or variable-length records grouped together into blocks.

Line sequential: designed to be printed and to be used with other programs, such as editors. These files consist of variable-length lines delimited by carriage-control characters.

Note - If a sequential file is opened by a process even without using any kind of lock, no other process can lock it in exclusive mode, unless the Java property *sun.nio.ch.disableSystemWideOverlappingFileLockCheck* is set to true.

Relative Files

Relative files are generally used to store data where low overhead is required. Records are available by record number that represents the record location relative to where the file begins. For example, the first record in the file has a relative record number of 1, the tenth record has a relative record number of 10, and so forth. The records can only have fixed length.

Note - If a relative file is opened by a process even without using any kind of lock, no other process can lock it in exclusive mode, unless the Java property *sun.nio.ch.disableSystemWideOverlappingFileLockCheck* is set to true.

Indexed Files

Indexed files are file with an index that allows easy random access to any record given its file key. isCOBOL natively supports two kind of indexed files: JISAM and c-tree. A File Connector solution is available to access c-tree, Acucobol-GT (Vision) and Micro Focus files.

Comparison between JISAM and c-treeRTG

The table below shows the differences between JISAM and c-tree.

The objective of this comparison is to help the user in choosing the right file system depending on his needs.

	JISAM	c-tree
maximum file size	9 EB	16 EB
maximum record size	2 GB	64 KB for fixed record length 2 GB for variable record length
maximum number of keys	no limit	no limit [A]
maximum key size	255 bytes	no limit
max number of segments per key	16	no limit [B]
maximum number of records	no limit	no limit
variable length records	not supported [C]	supported
transactions	not supported	supported
alternate collating sequence	supported	supported
OPEN INPUT WITH LOCK	not supported	supported
data encryption	supported[D]	supported
file compression	not supported	supported through configuration [E]
ODBC and JDBC access	supported via isCOBOL UDBC (separate product)	supported [F]
ODBC compliance	2.0	3.52
ADO.NET, PHP and Python	not supported	supported [F]
native dependences	no	yes
file handling utility	JUTIL	ctutil
monitor and tuning utilities	none	c-treeACEMonitor c-treeGauges c-treeISAMExplorer c-treeLoadTest c-treeLogAnalyzer c-treePerfMon c-treeTPCATest DrCtree
backup features	none	integrated online backup
data replication	not supported	supported via c-tree Replication Agent (separate product)

	JISAM	c-tree
memory files	not supported	supported
file pool	not supported	supported

[A] By default the maximum number of keys is 32, but it can be increased by setting MAX_DAT_KEY in the server configuration.

[B] By default the maximum number of segments is 16, but it can be increased by setting MAX_KEY_SEG in the server configuration.

[C] Variable length records are treated as fixed length records using the maximum record size.

[D] Encrypted JISAM files can't be read via ODBC and JDBC.

[E] The compression is activated by the configuration properties iscobol.file.index.datacompress (boolean) and iscobol.file.index.keycompress (boolean). The WITH COMPRESSION clause in the SELECT statement is ignored.

[F] The SQL Engine requires a specific license. With the standard license SQL features are available only for three hours from the c-tree Server startup.

JISAM

Overview

JISAM is a 100% Java-based indexed sequential access (ISAM) file system that runs on a wide range of platforms, from mainframes to handheld mobile devices. Now your business can deliver fast and efficient access to COBOL applications with ISAM data files anywhere Java technology runs, without the overhead of a relational database or investing in complex program change.

Key details

- Supplied with isCOBOL
- Written entirely in Java, so it runs anywhere, even on a mobile phone
- JUTIL utility provided for basic file management
- ISMIGRATE (Index File Migration) utility provided for one-step migration of data files from other data sources

Technical characteristics

The JISAM file system has the following characteristics:

- Maximum file size: 9 EB
- Maximum number of keys: no limit
- Maximum number of records: no limit
- Maximum key length: 255 bytes
- Maximum number of segments per key: 16
- Maximum record length: 2 GB

Currently JISAM has the following limitations:

- transactions are not supported

- native compression is not supported
- variable length records are not supported, the maximum record size is always used

Versions

The JISAM file system included in isCOBOL is version 2.

isCOBOL 2009 and previous versions support an old version of JISAM, version 1. If you need to share data with old versions isCOBOL, you can instruct the program to create JISAM 1 files by setting the following property in the configuration:

```
iscobol.jisam.version=1
```

Be aware that JISAM 1 has more limitations and does not perform as well as JISAM 2, so the above setting should be used only if necessary.

The table below lists the differences between the two versions of JISAM..

	JISAM 1	JISAM 2
maximum file size	2 GB	9 EB
maximum record size	32 KB	2 GB
maximum number of keys	no limit	no limit
maximum key size	255 bytes	255 bytes
max number of segments per key	8	16
maximum number of records	no limit	no limit

Lock behaviors

Java SE 6 throws an OverlappingFileLockException if an application attempts to lock a region that overlaps a region locked through another FileChannel instance. Previous versions did not check for file locks obtained by other FileChannel instances. Java SE 6 has added the system property sun.nio.ch.disableSystemWideOverlappingFileLockCheck to control java.nio.channels.FileChannel.lock file checking behavior.

When using Java SE 6 or higher, this property must be set to true, otherwise an unexpected lock condition occurs when a JISAM file is open by two programs in the same runtime session. The following snippet shows how to launch a COBOL program with the necessary setting:

```
iscrun -J-Dsun.nio.ch.disableSystemWideOverlappingFileLockCheck=true ProgramName
```

In addition to the above setting, the lock behavior can be configured through the following Framework Properties:

- iscobol.jisam.autolock_allowed (boolean)
- iscobol.file.index.lock_read_anyhow (boolean) *
- iscobol.file.index.lock_wait (boolean) *
- iscobol.file.index.read_lock_test (boolean) *

These properties have the same effect regardless of the Java version.

On Windows, when a lock is interrupted, it is kept for a while by the system. In thin client environment this behavior causes that, if a client is interrupted during a locking operation, the other clients get a *file locked* error.

Physical files

A JISAM file is always identified by two disc files.

The first file stores the key information and by default has the extension “idx”. You can change this extension by setting the property iscobol.file.index.index_suffix *.

The second file stores the data and by default has the extension “dat”. You can change this extension by setting the property iscobol.file.index.data_suffix *.

Encryption

JISAM supports the encryption of the data file. The encryption is activated by the WITH ENCRYPTION clause in FILE-CONTROL.

An example of encrypted file:

```
select arc assign to "arc"
       organization indexed
       with encryption
       record arc-k.
```

The data is encrypted using a key provided through the configuration. The configuration property iscobol.file.encryption.key * must be set to a value different from spaces, otherwise a file mismatch error is raised. The encryption key can be up to 16 characters long.

The Blowfish algorithm is used to encrypt data.

If an encrypted file is opened for input or i-o with the wrong encryption key, then a file corrupt error is raised.

If the encryption key is not set in the configuration, opening an encrypted file produces a 9X status.

c-treeRTG

c-treeRTG is a robust file server provided by Faircom that can be easily interfaced by isCOBOL. For more details, please consult the dedicated book in this documentation: isCOBOL Evolve: c-tree RTG.

VisionJ

isCOBOL provides a 100% Java-based interface to the Vision file system: VisionJ.

You can read and edit existing files created by ACUCOBOL-GT as well as create new Vision files.

You can lock a file or a record against other isCOBOL runtime sessions or ACUCOBOL-GT runtime sessions.

Being 100% Java-based, the VisionJ allows you to use Vision files on a wide range of platforms, from mainframes to handheld mobile devices.

The following Vision file formats are supported:

- Vision 3
- Vision 4

- Vision 5
- Vision 6

Transactions are supported without logging.

Compressed files are supported, encrypted files are not.

Configuration

The VisionJ behavior is affected by the generic configuration properties for File Handling Configuration, such as iscobol.file.prefix and iscobol.file.suffix.

There are also some specific configuration properties that affect only VisionJ:

- iscobol.visionj.v_apply_signature
- iscobol.visionj.v_buffers_per_file *
- iscobol.visionj.v_compress_factor
- iscobol.visionj.v_force_open
- iscobol.visionj.v_index_block_percent
- iscobol.visionj.v_internal_locks
- iscobol.visionj.v_lock_method
- iscobol.visionj.v_locks_per_file
- iscobol.visionj.v_mark_read_corrupt
- iscobol.visionj.v_max_files *
- iscobol.visionj.v_open_strict
- iscobol.visionj.v_read_ahead (boolean) *
- iscobol.visionj.v_seg_size
- iscobol.visionj.v_strip_dot_extension (boolean)
- iscobol.visionj.v_version

File Connectors

Overview

isCOBOL provides a File Connector technology for file systems with native parts. File Connectors separate ISAM native access from java process.

It's particularly useful when working in Application Server environment. For every isCOBOL Server thread (a thread is created for each connected Client) a separate native process is invoked. isCOBOL Server communicates with the native process through unnamed pipe.

The advantage of this approach it is to separate java code from native code to have a 100% pure java server.

This feature is currently supported for

- Acucobol-GT ISAM files (Vision)
- c-tree ISAM files
- DBMaker DCI
- Micro Focus ISAM files
- RM/COBOL ISAM files

The c-tree File Connector

The c-tree File Connector allows you to work on c-tree files managed by c-tree by separating ISAM native access from the java process.

Note - The ctreej file handler is preferable: it's thread safe and provides better performance than fscsc. You should consider using fscsc only if you don't want the JVM behind your application (or your application server) to load native libraries directly for some reason.

The File Connector executable is provided along with isCOBOL:

Platform	Executable
Linux/ Unix	$ISCOBOL/bin/**fscsc**
Windows	%ISCOBOL%\bin**fscsc.exe**

In order to make isCOBOL use the c-tree File Connector as file handler, the following setting must appear in the configuration:

```
iscobol.file.index=fscsc
```

The fscsc file handler runs the executable file fscsc. If this file is not in the system Path, you can specify its full name by setting the iscobol.file.connector.program configuration property.

The executable file fscsc must be of the same architecture as the c-tree client library, but it can be of a different architecture than the isCOBOL Framework. For example, the following combinations are both valid:

- 64-bit isCOBOL + 64-bit fscsc.exe + 64-bit ctree.dll
- 64-bit isCOBOL + 32-bit fscsc.exe + 32-bit ctree.dll

The following combination is not valid, instead:

- 64-bit isCOBOL + 64-bit fscsc.exe + 32-bit ctree.dll

The c-tree File Connector reads only the configuration provided by CTREE_CONF. See Configuring the client through CTREE_CONF for details.

The DCI File Connector

The DCI File Connector allows you to work on DBMaker by separating ISAM native access from the java process.

The File Connector executable is provided along with isCOBOL:

Platform	Executable
Linux/ Unix	$ISCOBOL/bin/**dcic**
Windows	%ISCOBOL%\bin**dcic.exe**

In order to make isCOBOL use the DCI File Connector as file handler, the following setting must appear in the configuration:

```
iscobol.file.index=dcic
```

The dcic file handler runs the executable file dcic. If this file is not in the system Path, you can specify its full name by setting the iscobol.file.connector.program.dcic configuration property.

The DCI connector library (dcic.dll on Windows and libdcic.so on Linux) is required in the library path (%PATH% on Windows and $LD_LIBRARY_PATH on Linux).

Note that this library is available only since version 5.4.2.

To retrieve the necessary libraries

1. browse to http://www.dbmaker.com/downloads.html,
2. select the desired DBMaker version from the bulleted list,
3. scroll down to find the section "DCI Downloads",
4. click on the download button to download the zip,
5. when the download is completed, open the zip and find the folder that matches your platform and open it,
6. find the "iscbl" subfolder and extract the dcic library (dcic.dll for Windows, libdcic.so for Linux).

To have the DCI connector library automatically loaded, copy dcic.dll to the isCOBOL's bin directory on Windows and copy libdcic.so to the isCOBOL's native/lib directory on Linux.

The executable file dcic must be of the same architecture as the dcic and dmapi libraries, but it can be of a different architecture than the isCOBOL Framework. For example, the following combinations are both valid:

- 64-bit isCOBOL + 64-bit dcic.exe + 64-bit dcic.dll + 64-bit dmapi54.dll
- 64-bit isCOBOL + 32-bit dcic.exe + 32-bit dcic.dll + 32-bit dmapi54.dll

The following combination is not valid, instead:

- 64-bit isCOBOL + 64-bit dcic.exe + 32-bit dcic.dll + 32-bit dmapi54.dll

The DCI connector is the same as DCI in terms of configuration, library routines and troubleshooting. Refer to Basic Configuration, Library Routines and Troubleshooting in the DCI documentation for more information.

The Micro Focus File Connector

The Micro Focus File Connector allows you to work on Micro Focus indexed files by separating ISAM native access from the java process.

Contact Veryant for more information about this solution.

The RM/COBOL File Connector

The RM/COBOL File Connector allows you to work on RM/COBOL indexed files by separating ISAM native access from the java process.

The File Connector executable is provided along with isCOBOL:

Platform	Executable
Windows	%ISCOBOL%\bin**rmc.exe**

In order to make isCOBOL use the RM/COBOL File Connector as file handler, the following setting must appear in the configuration:

```
iscobol.file.index=rmc
```

The rmc file handler runs the executable file rmc. If this file is not in the system Path, you can specify its full name by setting the iscobol.file.connector.program.rmc configuration property.

The rmc executable has dependences to RMFM32.DLL and OFM32.DLL, and it needs an RM/COBOL license named license.vlt to work correctly.

The executable file rmc must be of the same architecture as the RM/COBOL libraries, but it can be of a different architecture than the isCOBOL Framework. For example, the following combinations are both valid:

- 64-bit isCOBOL + 64-bit rmc.exe + 64-bit RMFM32.DLL + 64-bit OFM32.DLL
- 64-bit isCOBOL + 32-bit rmc.exe + 32-bit RMFM32.DLL + 32-bit OFM32.DLL

The following combination is not valid, instead:

- 64-bit isCOBOL + 64-bit rmc.exe + 32-bit RMFM32.DLL + 32-bit OFM32.DLL

The Vision File Connector

The Vision File Connector allows you you to work on Acucobol-GT Vision files by separating ISAM native access from the java process.

Contact Veryant for more information about this solution.

DCI

The DBMaker's COBOL Interface (DCI) allows isCOBOL to work on the DBMaker RDBMS with a ISAM approach.

DCI is available for Windows and Linux platforms.

In order to associate the DCI file handler to your indexed files, the following setting must appear in the configuration:

```
iscobol.file.index=dci
```

It's possible to associate only specific files to DCI. For example, the following configuration uses JIsam for all indexed files, except for "file1" that is associated to DCI:

```
iscobol.file.index=jisam
iscobol.file.index.file1=dci
```

Indexed files can be associated to DCI also through the CLASS clause in the SELECT statement. E.g.

```
           SELECT FILE1 ASSIGN TO FILE1-PATH
              ORGANIZATION INDEXED
              CLASS "com.iscobol.io.DynamicDCI"
              ACCESS MODE DYNAMIC
              RECORD KEY FILE1-KEY
              STATUS FILE-STATUS.
```

The DCI library (dci.dll on Windows and libdci.so on Linux) is required in the library path (%PATH% on Windows and $LD_LIBRARY_PATH on Linux).

To retrieve the necessary libraries

1. browse to http://www.dbmaker.com/downloads.html,
2. select the desired DBMaker version from the bulleted list,

3. scroll down to find the section "DCI Downloads",
4. click on the download button to download the zip,
5. when the download is completed, open the zip and find the folder that matches your platform and open it,
6. find the "iscbl" subfolder and extract the desired dci library.

The proper library depends on the DBMaker version and on the isCOBOL version:

- until DBMaker version 5.4.2 you use the dci library (dci.dll on Windows, libdci.so on Linux) for isCOBOL 2015 R1 and previous, and the dci_2016 library (dci_2016.dll on Windows, libdci_2016.so on Linux) renamed to dci for isCOBOL 2016 R1 and later.
- since DBMaker version 5.4.3 instead you use the dci library (dci.dll on Windows, libdci.so on Linux) for isCOBOL 2016 R1 and later, and the dci_2015 library (dci_2015.dll on Windows, libdci_2015.so on Linux) renamed to dci for isCOBOL 2015 R1 and previous.

Using an improper dci library may lead to memory access violation errors.

To have the DCI library automatically loaded, copy dci.dll to the isCOBOL's bin directory on Windows and copy libdci.so to the isCOBOL's native/lib directory on Linux.

Note - the DCI library is not thread safe. If you wish to use DCI in a application server environment like Tomcat or the isCOBOL Thin Client, then you should consider to rely on the The DCI File Connector.

Basic Configuration

DCI requires EFD dictionaries in order to manage COBOL indexed files as DBMaker database tables. The dictionaries are generated by the Compiler if the -efd option is used.

DCI looks for the configuration file pointed by the environment variable DCI_CONFIG. The basic configuration is:

Configuration Entry	Meaning
DCI_DATABASE	Name of the DBMaker database to connect
DCI_LOGIN	User name credential
DCI_PASSWD	Password credential
DCI_EFDPATH	Directory where EFD dictionaries are stored.

Example - the following configuration allows to connect to the default database DBSAMPLE5 with user SYSADM without password having the EFD dictionaries stored in the folder C:\myapp\efd:

```
DCI_DATABASE DBSAMPLE5
DCI_LOGIN SYSADM
DCI_EFDPATH C:\myapp\efd
```

Library Routines

Refer to DCI for the list of supported DCI library routines.

Troubleshooting

All the errors that can be mapped to a COBOL file status are mapped to a COBOL file status.

Other errors are returned as secondary code of the file status 9D.

The following table lists the most common ones:

Error code	Description
03	EFD dictionary not found.
5510	Invalid DCI connection handle.
5515	Mismatch between EFD and table structure.

Refer to the DBMaker Error Reference and Message manual for the complete list of error codes.

Where to go next

Refer to the DCI Manual from Casemaker for more information about the usage of DCI, including all the available configuration entries and library routines.

Btrieve

Btrieve is a transactional database software product based on Indexed Sequential Access Method (ISAM).

isCOBOL is able to interface Btrieve on Windows.

In order to associate the Btrieve file handler to your indexed files, the following setting must appear in the configuration:

```
iscobol.file.index=btrieve
```

It's possible to associate only specific files to Btrieve. For example, the following configuration uses JIsam for all indexed files, except for "file1" that is associated to Btrieve:

```
iscobol.file.index=jisam
iscobol.file.index.file1=btrieve
```

Indexed files can be associated to Btrieve also through the CLASS clause in the SELECT statement. E.g.

```
           SELECT BTR1 ASSIGN TO BTR1-PATH
              ORGANIZATION INDEXED
              CLASS "com.iscobol.io.DynamicBtrieve"
              ACCESS MODE DYNAMIC
              RECORD KEY BTR1-KEY
              STATUS BTR1-STATUS.
```

The wbtrv32 library (wbtrv32.dll) is required in the PATH.

The following features are not supported:

- Alternate collating sequence
- START WITH SIZE
- Transactions

Refer to the Pervasive Documentation for more information about the usage of Btrieve.

File Handlers

The isCOBOL Framework includes a series of file handlers that allows you to manage sequential, relative and indexed files. These file handlers are implemented by dedicated classes in the package com.iscobol.io. For most of them, an alias is provided in order to easily reference the file handler without specifying the full class name. The table below lists the available file handlers providing class name, alias and a brief description:

Indexed file handlers

Class-Name	Alias	Description
com.iscobol.extfh.ExtfhIndex	n/a	The file is handled by the 32bit EXTFH interface.
com.iscobol.extfh3.ExtfhIndex	n/a	The file is handled by the 64bit EXTFH interface.
com.iscobol.io.DynamicBtrieve	btrieve	The file is handled by the Btrieve interface.
com.iscobol.io.DynamicConnector	fscsc	The file is handled by c-tree through FileConnector.
com.iscobol.io.DynamicCtree	ctree	The file is handled by c-tree of version 9.5.51961 or previous.
com.iscobol.io.DynamicCtree2	ctree2	The file is handled by c-tree of version 9.5.53702 or later.
com.iscobol.io.DynamicCtreeJ	ctreej	The file is handled by c-tree of version 10.4.0.39701 or later through a Java interface. Note that the native client library is always required.
com.iscobol.io.DynamicDCI	dci	The file is handled by the DBMaker DCI interface.
com.iscobol.io.DynamicDConnector	dcic	The file is handled by the DCI File Connector.
com.iscobol.io.DynamicEasyDB	easydb	The file is handled by isCOBOL DatabaseBridge.
com.iscobol.io.DynamicJIsam	jisam	The file is a JIsam archivie (default).
com.iscobol.io.DynamicMConnector	mfc	The file is handled by the Micro Focus File Connector.
com.iscobol.io.DynamicRConnector	rmc	The file is handled by the RM/COBOL File Connector.
com.iscobol.io.DynamicRemote	remote	The file is handled remotely by the file server feature provided by isCOBOL Server.
com.iscobol.io.DynamicVConnector	vfc	The file is handled by the Vision File Connector.
com.iscobol.io.DynamicVision	vision	(deprecated) The file is handled by the Acucobol-GT Vision interface. The runcbl library, provided separately, is required.
com.iscobol.io.DynamicVisionJ	visionj	The file is handled by a 100% Java-based Vision interface. No native components are required.
com.iscobol.io.ScanMF	n/a	The file is handled by an internal Micro Focus files interface. You can only sequentially read next in input mode, other i-o is not supported. Micro Focus formats 1, 2, 3, 4 and 8 as well as C-ISAM are supported. Compressed files and split files are not supported, they won't be opened by the ScanMF class. A parsing error may occur while reading files with variable length records if the area of a record was reused by the MF runtime, e.g. due to a REWRITE statement or because it overwrote the area of a deleted record in order to save space.

Class-Name	Alias	Description
com.iscobol.io.ScanRMKF	n/a	The file is handled by an internal RM/COBOL files interface. You can only sequentially read next in input mode, other i-o is not supported.
com.iscobol.io.ScanVision	n/a	The file is handled by an internal Vision interface. You can only sequentially read next, other i-o is not supported. Vision formats 3 to 6 are supported. The following file types are currently not supported: - encrypted files - Vision files version 1 and 2 A 9D error is returned on open of unsupported files.

Binary Sequential file handlers

Class-Name	Alias	Effect
com.iscobol.extfh.ExtfhSequential	n/a	The sequential file is handled by the 32bit EXTFH interface.
com.iscobol.extfh3.ExtfhSequential	n/a	The sequential file is handled by the 64bit EXTFH interface.
com.iscobol.io.DynamicEasyDBSeq	easydb	The file is handled by isCOBOL DatabaseBridge.
com.iscobol.io.DynamicMFSequential	mfsequential	The file is handled in compatibility with Micro Focus.
com.iscobol.io.DynamicRemote	remote	The file is handled remotely by the file server feature provided by isCOBOL Server.
com.iscobol.io.DynamicSeqACU	seqacu	The fixed length sequential file is handled in compatibility with Acucobol.
com.iscobol.io.DynamicSequential	sequential	The file is a binary file on the local machine (default).
com.iscobol.io.DynamicVarSeqACU	varseqacu	The variable length sequential file is handled in compatibility with Acucobol.
com.iscobol.io.RemoteRelative	remote	The file is a binary file on the client machine (Application Server).

Relative file handlers

Class-Name	Alias	Effect
com.iscobol.extfh.ExtfhRelative	n/a	The relative file is handled by the 32bit EXTFH interface.
com.iscobol.extfh3.ExtfhRelative	n/a	The relative file is handled by the 64bit EXTFH interface.
com.iscobol.io.DynamicEasyDBRel	easydb	The file is handled by isCOBOL DatabaseBridge.
com.iscobol.io.DynamicRelative	relative	The file is a binary file on the local machine (default).
com.iscobol.io.DynamicRemote	remote	The file is handled remotely by the file server feature provided by isCOBOL Server.

Line Sequential file handlers

Class-Name	Alias	Effect
com.iscobol.extfh.ExtfhLineSequential	n/a	The file is handled by the 32bit EXTFH interface.
com.iscobol.extfh3.ExtfhLineSequential	n/a	The file is handled by the 64bit EXTFH interface.
com.iscobol.io.DynamicEasyDBLSeq	easydb	The file is handled by isCOBOL DatabaseBridge.

Class-Name	Alias	Effect
com.iscobol.io.DynamicLSeq8bit	lseq8bit	The file is handled by the standard isCOBOL interface (default).
com.iscobol.io.DynamicLSeqACU	lseqacu	The file is handled by an Acucobol compatible interface that discards form feed, carriage return and line feed before returning the record.
com.iscobol.io.DynamicLSeqMF_N	lseqmf_n	The file is handled by a Micro Focus compatible interface that doesn't treat 0x0A in COMP fields as line feed.
com.iscobol.io.DynamicRemote	remote	The file is handled remotely by the file server feature provided by isCOBOL Server. It doesn't work for stream files and print files. It doesn't work if program is compiled with the -flsu option.

It's also possible to create your own file handler by extending an existing one or by implementing it from scratch. A couple of examples are installed along with isCOBOL under the folder sample/data-access/files-redirect.

Chapter 7

Programming Guides

This book contains instructions for the interoperability between isCOBOL and external resources like RDBMS, the Apache Ant software tool and file system supporting the ExtFH interface. It also provides guidelines to modernize and improve your COBOL application.

Apache Ant integration

Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. Ant can also be used effectively to build non Java applications.

In this chapter we explain how to use Ant to build and run an isCOBOL application.

Basic concepts

Before you read, you should get familiar with these concepts:

Buildfile	Apache Ant's buildfiles are written in XML. Each buildfile contains one project and at least one (default) target. Targets contain task elements. Each task element of the buildfile can have an id attribute and can later be referred to by the value supplied to this. The value has to be unique.
Project	Each project defines one or more targets. A target is a set of tasks you want to be executed. When starting Ant, you can select which target (or targets) you want to have executed. When no target is given, the project's *default* is used.
Target	A target is a container of tasks that cooperate to reach a desired state during the build process. Targets can depend on other targets and Apache Ant ensures that these other targets have been executed before the current target. For example you might have a target for compiling and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target depends on the compile target.
Task	A task is a piece of code that can be executed. There is a set of built-in tasks, but it is also very easy to write your own. Below you will find information about how to create a task that executes the isCOBOL Compiler.
FileSet	A FileSet is a group of files. These files can be found in a directory tree starting in a base directory and are matched by patterns

Compiling

In order to compile COBOL programs using Ant, the following task must be defined in the build file:

```
<taskdef name="iscc" classname="com.iscobol.ant.iscc"/>
```

The class indicated in this task is included in the iscobol.jar library. This library must be available in the Classpath when the task runs.

The iscc task supports the following attributes:

Attribute	Meaning	Default value
javac	Path to an external javac compiler to be used	
javacOptions	Java complier options to be used, i.e. -classpath to point to the iscobol.jar library where com.iscobol.ant.iscc is stored	
nosummary	Set it to "false" to have the Compiler outcome printed on the console	"true"
nowarn	Set it to "true" to suppress Compiler warnings	"false"
noerr	Set it to "true" to suppress Compiler errors. Note that Severe errors will not be suppressed anyway	"false"
force	Set it to "true" to compile programs even if they're not out of date	"false"
options	isCOBOL Compiler options, i.e. "-cm -dcm" See Compiler Options for the list of all the available options. You should not use the "-od" option, but set the destDir attribute instead	"-jc"
FailOnError	Set it to "false" to continue the build process even if a Severe error occurs. By default, the build process stops in this case	"true"
destDir	Path where the compiled class files will be stored	

The iscc task will include one or more FileSet elements that tell which source files must be compiled.

The following snippet shows how to compile the HELLO-WORLD.cbl program in debug mode:

```
<iscc
     javacOptions="-classpath ${iscobol-classpath-prop} "
     nosummary="false"
     nowarn="true"
     noerr="true"
     force="true"
     options="-d"
     failOnError="true"
     destDir="${build.dir}">
    <fileset dir="${src.dir}">
       <include name="HELLO-WORLD.cbl"/>
    </fileset>
</iscc>
```

Running

In order to run or debug a COBOL program using Ant, you can configure a Java build-in task to run the isCOBOL Runtime class (com.iscobol.invoke.Isrun), e.g.

```
<java classname="com.iscobol.invoke.Isrun"
      fork="true"
      classpath="${iscobol-classpath-prop}" >
      <arg value="-d" /> <!-- remove this arg to run without Debugger -->
      <arg value="HELLO_WORLD" />
</java>
```

Example

An example of Apache Ant integration is provided along with the isCOBOL SDK.

You can find a build.xml file under the sample/issamples directory.

Having Apache Ant installed in the system, you can change to that directory and issue the command:

```
ant compile
```

It will recompile all the sample programs in this directory.

In order to run the sample container using Ant, issue the command:

```
ant run
```

For more information, setup files and install instructions for Apache Ant, visit the official web site https://ant.apache.org.

JDBC

JDBC (short for Java DataBase Connectivity) is a Java API that enables Java programs to execute SQL statements. This allows Java programs to interact with any SQL-compliant database. Since nearly all relational database management systems (RDBMS) support SQL, and because Java itself runs on most platforms, JDBC makes it possible to write a single database application that can run on different platforms and interact with different RDBMS.

isCOBOL provides two ways to interact with RDMBS

- Database Bridge
- ESQL syntax

Both of them take advantage of JDBC.

In order to let isCOBOL interact with a RDBMS you need the proper JDBC driver. JDBC drivers are Java libraries (jar) that are usually supplied by the RDBMS vendor. Each RDMBS has its own drivers. The Java library must appear in the CLASSPATH and the driver class name must be specified by the iscobol.jdbc.driver configuration property.

In addition you must specify the connection url by setting the iscobol.jdbc.url property.

Once driver and url have been set, your program is able to connect to the RDBMS through JDBC.

Common Driver and URL settings

Driver and url settings are usually provided by database vendors through documentation and sample programs. For your convenience, this manual lists the settings for the most common RDBMS.

Oracle

Library:

```
ojdbc8.jar
```

Value for iscobol.jdbc.driver:

```
oracle.jdbc.OracleDriver
```

Value for iscobol.jdbc.url:

```
jdbc:oracle:thin:<Username>/<Password>@<ServerName>:<Port>:<Sid>
```

Value for iscobol.jdbc.url when using a TNS:

```
jdbc:oracle:thin:<Username>/
<Password>@(description=(address=(host=<ServerName>)(protocol=tcp)(port=<Port>))(conne
ct_data=(sid=<Sid>)))
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

Microsoft Sql Server

Library:

```
sqljdbc4.jar
```

Value for iscobol.jdbc.driver:

```
com.microsoft.sqlserver.jdbc.SQLServerDriver
```

Value for iscobol.jdbc.url:

```
jdbc:sqlserver://
<ServerName>:<Port>;user=<UserName>;password=<Password>;encrypt=false;DatabaseName=<Da
tabaseName>
```

NOTE - A third party pure java JDBC driver is also available for Microsoft Sql Server. Its name is JTDS and it can be found at the following website: http://jtds.sourceforge.net.

Library:

```
jtds-1.2.5.jar
```

Value for iscobol.jdbc.driver:

```
net.sourceforge.jtds.jdbc.Driver
```

Value for iscobol.jdbc.url:

```
jdbc:jtds:sqlserver://<ServerName>:<Port>/
<DatabaseName>;user=<UserName>;password=<Password>
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

IBM DB2

Library:

```
db2jcc4.jar
```

Value for iscobol.jdbc.driver:

```
com.ibm.db2.jcc.DB2Driver
```

Value for iscobol.jdbc.url:

```
jdbc:db2://<ServerName>:<Port>/<DataBaseName>:user=<UserName>;password=<Password>;
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

Informix

Library:

```
ifxjdbc.jar
```

Value for iscobol.jdbc.driver:

```
com.informix.jdbc.IfxDriver
```

Value for iscobol.jdbc.url:

```
jdbc:informix-sqli://<ServerName>:<Port>/
<DatabaseName>:informixserver=<Instance>;user=<UserName>;password=<Password>
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

MySQL and MariaDB

MySQL

Library:

```
mysql-connector-java-<version>-bin.jar
```

Value for iscobol.jdbc.driver:

```
com.mysql.jdbc.Driver
```

Value for iscobol.jdbc.url:

```
jdbc:mysql://<ServerName>:<Port>/<DatabaseName>?user=<UserName>&password=<Password>
```

MariaDB

Library:

```
mariadb-java-client-<version>.jar
```

Value for iscobol.jdbc.driver:

```
org.mariadb.jdbc.Driver
```

Value for iscobol.jdbc.url:

```
jdbc:mariadb://<ServerName>:<Port>/<DatabaseName>?user=<UserName>&password=<Password>
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

PostgreSQL

Library:

```
postgresql-<version>.jar
```

Note - the above driver is the one certified with Java 1.8. Previous Java versions require a different library. Consult the Postgres documentation for details.

Value for iscobol.jdbc.driver:

```
org.postgresql.Driver
```

Value for iscobol.jdbc.url:

```
jdbc:postgresql://<ServerName>:<Port>/
<DatabaseName>?user=<UserName>&password=<Password>
```

See DatabaseBridge and JDBC/ESQL Configuration for other JDBC settings that affects the post-connection behaviors, like autocommit.

Common JDBC connection errors

```
no suitable driver found for <url>
```

This error is returned when the JDBC Driver cannot resolve the connection URL specified by iscobol.jdbc.url property. Double check the syntax of the URL and ensure that it follows the documented specifications.

ExtFH

The Callable File Handler, ExtFH, is a loadable file handling subsystem with an open architecture developed by Micro Focus. It can be used independently with a variety of programming languages, enabling you to create powerful file processing tools, as well as sophisticated database applications.

The Callable File Handler (ExtFH) enables programs to perform complex file operations that are not usually directly supported by your language syntax.

The main advantages provided by ExtFH are:

- Fast platform-independent file handling.
- Access to COBOL files by non-COBOL languages.

The ExtFH interface provides a way for applications to transparently access a file system such as DB2 and Oracle for record storage. In transaction processing environments, ExtFH is typically used to handle data access for batch programs.

You can also write your own file handler and run it in place of ExtFH, as long as it conforms to the interface defined by Micro Focus.

Using ExtFH from isCOBOL

isCOBOL's file handling mechanism is based on a pluggable architecture that allows any custom file handlers as described below.

isCOBOL also supplies a special set of classes that can interface any file handler compliant with the Micro Focus ExtFH interface, both 32 and 64 bit.

The EXTFH implementation must reside in a shared library, whose name is, by default, EXTFH (its file name will be libEXTFH.so on Unix-like systems and EXTFH.DLL on Windows). The name of the library can be changed by setting the following property in the configuration:

```
iscobol.extfh.libname=MyExtfhLib
```

The library must contain a function named EXTFH and must be built by the user linking the appropriate libraries for file handling with the Java native interface com_iscobol_extfh_EXTFH.o provided with isCOBOL.

For your convenience a Visual C project is installed on Windows along with isCOBOL, while a Makefile is provided for the other platforms. They are a good starting point to perform the link.

Along with the Java interface, another C module named xfhname is provided, both in source and object format. By modifying xfhname source code it is possible to change the name of the function EXTFH in order to avoid name conflicts.

```
extern void EXTFH (char* op, char* fcd);
void (*extfhFunction)(char* op, char* fcd) = EXTFH;
```

ExtFH is a single interface for any type of file and, by default, it fully substitutes the standard file handler. isCOBOL permits the use of the ExtFH interface for each file type. Twelve classes are provided (six for 32 bit ExtFH and six for 64 bit ExtFH). The following settings must appear in the configuration in order to make isCOBOL use ExtFH for the specific file types.

(for 32 bit ExtFH)

```
iscobol.file.index=com.iscobol.extfh.ExtfhIndex
iscobol.file.relative=com.iscobol.extfh.ExtfhRelative
iscobol.file.sequential=com.iscobol.extfh.ExtfhSequential
iscobol.file.linesequential=com.iscobol.extfh.ExtfhLineSequential
iscobol.file.input=com.iscobol.extfh.ExtfhInput
iscobol.file.output=com.iscobol.extfh.ExtfhOutput
```

(for 64 bit ExtFH)

```
iscobol.file.index=com.iscobol.extfh3.ExtfhIndex
iscobol.file.relative=com.iscobol.extfh3.ExtfhRelative
iscobol.file.sequential=com.iscobol.extfh3.ExtfhSequential
iscobol.file.linesequential=com.iscobol.extfh3.ExtfhLineSequential
iscobol.file.input=com.iscobol.extfh3.ExtfhInput
iscobol.file.output=com.iscobol.extfh3.ExtfhOutput
```

A new C function has been implemented to help determine which files must be managed by ExtFH. It is an extension to ExtFH which can be used at run-time, that accepts a file name parameter and returns the file's organization. The prototype is shown below:

```
int isFileHandledByExtfh (char *path, int fileType);
```

Parameters:

path	the name of the file; this variable is 256 byes long and can be changed by the function; in this case the modified name is passed back to the standard isCOBOL file manager, independently from the value returned by the function.

fileType	an integer whose content indicates the file type according to the following table: 1 TYPE_INDEX 2 TYPE_RELATIVE 3 TYPE_SEQUENTIAL 4 TYPE_LINE_SEQUENTIAL 5 TYPE_OUTPUT 6 TYPE_INPUT

If this function returns 0 it means that this file must be handled by isCOBOL while any other returned value means that this file must be handled through the EXTFH interface.

In the module xfhname a stub function that always returns 1 is supplied in order to work when the above extension is not required, so if this function is implemented with the EXTFH function, the stub must be removed.

If the property iscobol.extfh.intrinsic_file_manager (boolean) is set in the configuration, then the isFileHandledByExtfh function is invoked each time a file is opened. If it returns 0 then the isCOBOL file manager is called according to the standard configuration properties, otherwise the file will be handled through the EXTFH interface.

Accessing isCOBOL files from other languages through ExtFH

The ExtFH interface can also be used to access isCOBOL standard files from other programming languages.

The iscobolc library provided with isCOBOL exports the EXTFH function.

Other programming languages can access isCOBOL files by calling this function. There are two methods supporting this access:

- call EXTFH function that is exported by iscobolc dynamic library (iscoboc.dll on Windows platforms and libiscobolc.so on Unix platforms). (dynamic approach)
- link the iscobolc import library (iscobolc.lib on Windows and libiscobolc.a on Unix) into your program and then call the EXTFH function. (static approach)

Syntax

```
EXTFH(op_code,&fcd);
```

Parameters

op_code	hexadecimal byte specifying the operation code. The op_code 0006 (GetFileInfo) has the following limitations: • it works only for indexed files; • the 'file dimension' returned is actually the number of active records multiplied the maximum record length; • no information about 'sparse' nor 'compression' is returned.
fcd	(File Control Description) 100-byte area that contains information about the file in use. The calling program must complete the appropriate fields in the FCD before calling ExtFH. After performing the specified operation, ExtFH completes the appropriate fields in the FCD before passing control back to the calling program.

Please refer to the ExtFH specifications for details about the two parameters.

ExtSM

The Callable Sort Module (ExtSM) is a standalone sort routine that enables you to sort and re-order data files. Its call interface enables you to take advantage of alternative sort modules that are usually faster than the default run-time system sort module and provide greater flexibility in sorting data.

The main advantages provided by ExtSM are:

- Use either quick sort or insertion sort techniques
- Choose ascending or descending key ordering
- Use an externally supplied collating sequence

Using ExtSM from isCOBOL

The ExtSM implementation must reside in a shared library, whose name is EXTSM (its file name will be libEXTSM.so on Unix-like systems and EXTSM.DLL on Windows).

You can make the runtime invoke ExtSM, instead of the internal sort module, for quicker execution of COBOL SORT and MERGE syntax in your program.

To use ExtSM, the following setting must appear in the configuration.

(for 32 bit ExtSM)

```
iscobol.sort=com.iscobol.extfh.ExtsmSort
```

(for 64 bit ExtSM)

```
iscobol.sort=com.iscobol.extfh3.ExtsmSort
```

Audit

An information technology audit (or information systems audit) is an examination of the management controls within an Information technology (IT) infrastructure. The evaluation of obtained evidence determines if the information systems are safeguarding assets, maintaining data integrity, and operating effectively to achieve the organization's goals or objectives.

isCOBOL's audit feature builds on top of the file redirect technology and allows COBOL developers to add logging on all I/O operations without source code modifications.

Installed items

The necessary tools are installed along with the Isapplication sample in the sub directory *sample/isapplication* under the isCOBOL installation directory. The folder is structured as follows:

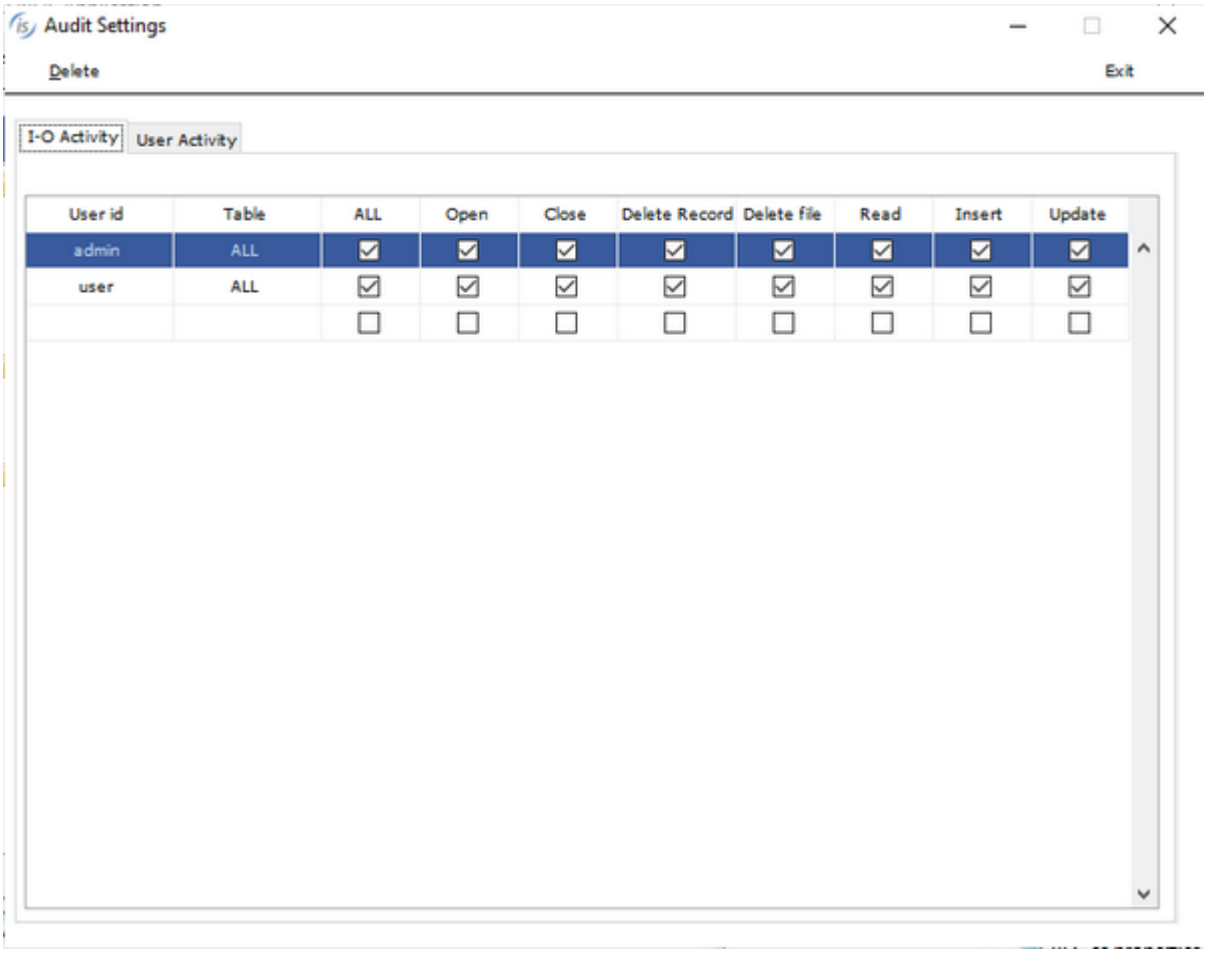
- The *copylib/audit* folder contains:
    - *audit-linkage.wrk*: Copy of Linkage Section for the AUDIT.cbl program. Programs that call AUDIT should include this copy in their Working-Storage Section.
    - *audit.wrk*: Copy of Working-Storage Section items, contains the audit variables
    - *auditlog.sl* and *auditlog.fd* describe the file that contains the audit information
    - *auditfilesettings.sl* and *auditfilesettings.fd* describe the file that contains the audit settings for file operations
    - *auditlogsettings.sl* and *auditlogsettings.fd* describe the file that contains the audit settings for user login/logout and the program execution information
- The *source/audit* folder contains the source code of the audit programs:
    - *AUDIT.cbl:* This program contains the audit function. Call it with with the appropriate op-code to execute audit function:
        - - start and stop the auditlog thread program
        - - pass the name of the logged user to the audit program
        - - load the audit settings from the file
        - - trace the user login and logout
        - - trace the start and the end of a program
    - *AUDITANALYSIS.cbl:* This program allows you to review the audit log in a graphical user interface.
    - *AUDITLOG.cbl*: This program writes the file's audit information into the *auditlog* file. Call this program in a thread at the startup of your program. The copybook called *audit.cpy* includes a paragraph that starts the audit thread and another paragraph that retrieves the information from the AuditTrigger program using the communication between threads.

- *AUDITSETTINGS.cbl*: This program allows you to configure the audit settings in a graphical user interface.
- *AuditTrigger.cbl*: The trigger program. This program checks the audit settings and sends the requested operation data thru a thread message to the AUDITLOG program. This COBOL program generates a lot of java classes. The classes of this kind of program must be placed into the Classpath. The code_prefix will not work for the trigger programs.

Follow the instructions in the README file to compile these COBOL programs and enable audit in the Isapplication's configuration.

Audit configuration

isCOBOL provides a graphical configurator for the audit feature. The program is named AUDITSETTINGS and can be launched from the Isapplication menu.



Using this tool you can easily activate or deactivate the logging of specific I/O operations and also choose which users will have their activity logged.

In the "I-O Activity" tab you can choose which file operation(s) to trace. Specify the name of the user as well as the name of the file. To trace all operation of a specific file click on "ALL" checkbox. If you want trace the operation on all file type "ALL" into the column "table"

The operations you can trace are:

Operation	Value stored into the auditlog file
Open	fo
Close	fc
Delete record	fd
Delete file	ff
Read (sequential and random)	fr
Write	fw
Rewrite	fx

In the "User Activity" tab you can choose to trace user access and program execution information. You will specify the name of the user.

The operations you can trace are:

Operation	Value stored into the auditlog file
Login	pi
Logout	po
Program start	ps
Program end	pe

How to use the audit feature

In order to activate the audit feature you create a separate thread that is responsible for logging the user activity on files according to the settings that you configured using the AUDITSETTINGS utility.

The necessary operations are provided by the AUDIT program. The program starts a thread that works along with your program and registers i-o operations to a log. The program's usage is explained below.

1. Add the audit-linkage.wrk copybook to the Working-Storage Section:

```
WORKING-STORAGE SECTION.
    copy "audit-linkage.wrk".
```

2. Start the audit thread:

```
    set audit-start-log to true
    call "AUDIT" using audit-link
```

3. Load the AUDIT settings into a table from the *auditfilesettings* file, e.g.:

```
move "MyUser"           to audit-user-logged
set audit-load-settings to true
call "AUDIT" using audit-link
```

4. Register the logon of the user according to the audit settings:

```
set audit-register-login   to true
call "AUDIT" using audit-link.
```

5. Register the start of the execution of a COBOL program according to the audit settings, e.g.:

```
move "MyProgramName" to audit-prg-to-launch
set audit-register-pgm-start  to true
call "AUDIT" using audit-link
```

6. Call your COBOL program:

```
call "MyProgramName"
```

7. Register the end of the execution of a COBOL program according to the audit settings:

```
set audit-register-pgm-end  to true
call "AUDIT" using audit-link
```

8. Register the logout of the user according to the audit settings:

```
set audit-register-logout   to true
call "AUDIT" using audit-link.
```

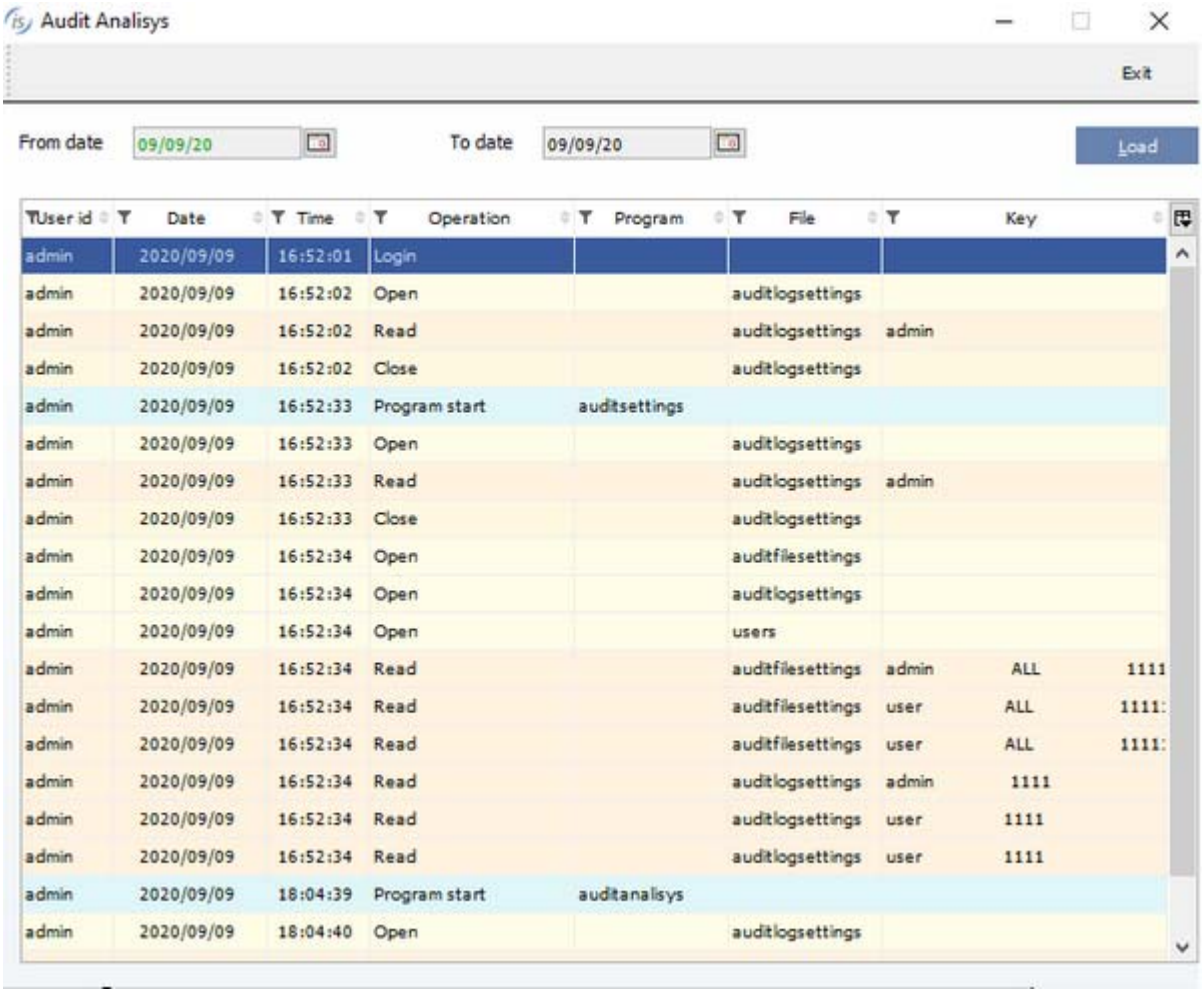
9. Stop the audit thread:

```
set audit-stop-log to true
call "AUDIT" using audit-link
```

In order to have i-o operations logged, files must be assigned to the AuditTrigger file handler. Refer to the README file in the Isapplication sample for more details and examples.

Audit log review

isCOBOL provides a graphical utility to review the audit log. The program is named AUDITANALYSIS and can be launched from the Isapplication menu.



Moving from Fat Client to Thin Client

This chapter provides guidelines for transitioning from a fat client environment to a thin client environment.

Concepts

In a fat client environment the server is used only for file hosting. The server hosts the data files, the program objects and possibly also the runtime executables, but the runtime process runs on every single client PC, so the client machines are responsible for the processing. Files are shared between the server and the clients via network share or file servers. This kind of architecture is not optimized and is more difficult to maintain.

In a thin client environment instead the server does most of the work. The server hosts data files, program objects and runtime executables and it's also responsible for the processing. Client machines only manage the user interface. Files are accessed locally on the server. This kind of architecture is more optimized and easier to maintain.

Why should you move from Fat Client to Thin Client?

There are some advantages in performing this change:

- better performance, as the elaboration is performed server-side and the server hardware is usually more powerful than the client PC hardware; also accessing data files locally on the server is faster than accessing the same files via network share or file server,
- memory saving, as class definitions and string constants are kept in memory on the server and shared between multiple instances of the program run by different clients,
- web enablement, as a thin client application can run in a web-browser via the WebClient technology (see isCOBOL Evolve: WebClient for more information).

Issues that you should be aware of

Some application concepts, like printing, have different rules between fat client and thin client. Below is a list of the most common differences between the two environments along with some advice on how to deal with these differences.

Printing

In a fat client environment the runtime sees only the printers installed on the client PC. In a thin client environment instead, the runtime sees both server printers and client printers. By default the COBOL application works with the client printers, but you can switch it on the server printers through the WINPRINT-SET-PRINTER-AS function.

User specific files

Some COBOL applications create files that are specific for the user that is running the application. Usually they're temporary files and they're created on the local drive when running in a fat client environment (e.g. "C:\Temp\work.tmp"). Every client has its own local drive so there is no conflict. After moving to a thin client environment, the local drive is on the server and therefore it's shared among all the connected clients. A little code change is required in this case: the file names must be made unique, for example by including the client machine name (or any other info that you can retrieve by calling the A$CURRENT_USER routine) in the path. For example, instead of having "/tmp/work.tmp", use "/tmp/client1/work.tmp", "/tmp/client2/work.tmp", etcetera.

Opening files with the associated application

The C$DESKTOP routine and the C$EASYOPEN routine open a file with the associated application on the same machine where the runtime is running. In a thin client environment this would cause the associated application to be launched on the server, without possibility for the user to interact with it. Set the *csFlag* parameter to 1 to open the file on the client machine.

Processing files stored on the client PC

The COBOL application may require the user to load a file stored on the client PC. Let's think for example about a CSV file that includes data to be imported in the application database. The COBOL application asks the user to provide the CSV file location via the OPENSAVE-OPEN-BOX function, then opens the CSV file for input, reads it and writes the data to the proper indexed data file. In a fat client environment this processing is completely performed on the local PC using client paths (e.g. read "C:\Downloads\orders_may_2019.csv" and write "K:\data\orders"). In a thin client environment instead, since the processing is performed server side, you need to transfer the CSV file to the server before reading it. It can be done via the C$COPY routine, e.g.

```
CALL "C$COPY" USING "@[display]:C:\Downloads\orders_may_2019.csv"
                    "/tmp/orders_may_2019.csv"
```

After the call to C$COPY, the COBOL application can transfer data (E.g. read "/tmp/orders_may_2019.csv" and write "/opt/myapp/data/orders").

The same consideration can be made for the opposite scenario, where the user is prompted to save a file via the C$PARAMSIZE function. The user will choose a client side path, but the file is on the server, so the program must download it to the client via the C$COPY routine, e.g.

```
CALL "C$COPY" USING "/tmp/download_me"
                    "@[display]:C:\Downloads\download_me"
```

Note - In a thin client environment the C$OPENSAVEBOX routine shows only client paths, it doesn't give the client access to paths on the server.

File case

This difference affects environments with a Linux/Unix server and Windows clients. In fat client the files are searched with Windows rules, so they're searched in a case insensitive way. In thin client instead, since files are accessed locally on the server, Unix rules apply and the file case matters. Let's make a practical example: we have a path on the server, "/opt/myapp/data", that the client PC sees as "K:\data" via Samba. In this path we have a file named "file1". In fat client the runtime can open successfully all these full names: "K:\data\file1", "K:\DATA\FILE1", "K:\Data\File1", etcetera. In thin client it must use exactly "/opt/myapp/data/file1" as the following full names would be invalid: "/opt/myapp/DATA/FILE1", "/opt/myapp/Data/File1", etcetera. The iscobol.file.case configuration entry may help in addressing this issue.

Library routines and external functions

Library routines that interact with the user interface or with printers automatically work on the client PC when running in a thin client environment. All the other routines, instead, work on the server by default. You can use the CALL CLIENT statement to use a specific routine on the client PC, for example:

```
*retrieve information about a file stored on the server
 CALL "C$FILEINFO" USING "/tmp/file1"
                         fileInfo.

*retrieve information about a file stored on the client
 CALL CLIENT "C$FILEINFO" USING "C:\Temp\file1"
                                fileInfo.
```

Refer to Library Routines for the list of routines that work on the client PC by default and the list of routines that can be called with a CALL CLIENT statement.

External C functions implemented in shared libraries are executed on the server machine by default when running in a thin client environment. You can use the CALL CLIENT statement to call a C function on the client PC, assuming that the shared library is installed on the client PC.

Ensure that C functions executed on the server are thread safe. If they're not thread safe, consider creating a separate task for each connected client by setting iscobol.as.multitasking to 1.

Java classes and Java-Beans

In a thin client environment, the user interface of a java-bean is displayed on the client PC while the backend processing is performed on the server. For this reason, the java-bean libraries must be installed on both client and server.

It is possible to invoke static methods of Java classes installed on the client PC, see callStaticMethod for details.

If you need to invoke non-static methods of Java classes installed on the client PC, create a COBOL program that invokes these methods, install the COBOL program on the client PC and call it with a CALL CLIENT statement.

Memory and CPU usage

In a thin client environment, the JVM behind the isCOBOL Server creates a new thread for each connected client. This makes the JVM use more memory and CPU than a JVM running the isCOBOL Framework in a stand-alone installation. Refer to Measuring the load of your COBOL application for advice about the tuning of the Server's JVM.

Additional issues introduced by WebClient

After moving from fat client to thin client, a possible next step is to let clients connect via web browser. The advantage of this solution is that neither Java nor isCOBOL need to be installed on the client machines, the user needs just a web browser.

This kind of solution is provided via WebClient. For more information about this product, read isCOBOL Evolve: WebClient.

There are some differences between a standard thin client environment and a WebClient environment. Read Known limitations and differences between WebClient and Thin Client for more information.

Retrieving the current runtime environment

Given the information provided in this chapter, it's important to know what's the current runtime environment in order to perform specific operations or not. It is possible to know if you're running in thin client by checking the IS-REMOTE condition in the TERMINAL-ABILITIES group item, for example:

```
ACCEPT TERMINAL-ABILITIES FROM TERMINAL-INFO.
IF IS-REMOTE
   | running in thin client
ELSE
   | not running in thin client
END-IF.
```

It is possible to know if you're running with WebClient by calling the C$GETRUNENV routine, for example:

```
CALL "C$GETRUNENV".
IF RETURN-CODE = RUNENV-WEB-CLIENT
  | running in WebClient
ELSE
  | not running in WebClient
END-IF.
```

Add the iscobol.def copybook to the Working-Storage for the definition of the items referenced in the above code snippets.

Keeping versions synchronized

It's important that the isCOBOL Client and isCOBOL Server are the same version, otherwise the connection fails with this error:

```
ERROR: Client release (n1) is incompatible with Application Server (n2)
```

You can consider to set up and Automatic Client update in order to keep the Client versions syncronized with the Server version.

Modernization Guides

UI Modernization (from character based to graphical and web)

This guide explains the modernization process of a legacy COBOL application with character based user interface. It discusses the necessary steps to transform the character based user interface into a graphical user interface and eventually to a web user interface.

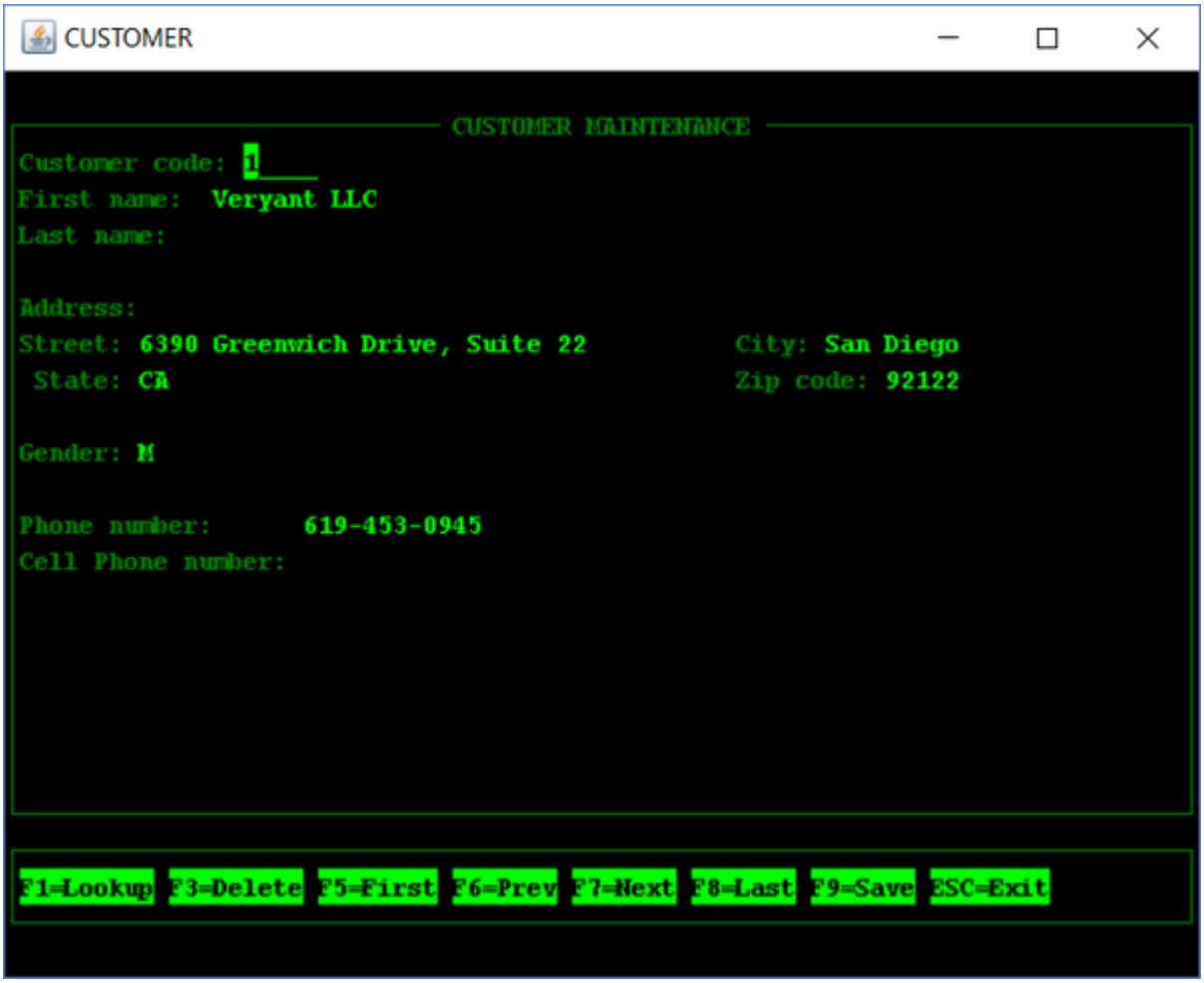
The code snippets used by this guide are taken from the modernization example installed with isCOBOL in the folder *sample/modernization*.

For an easy conversion of the user interface it's good practice to have the user interface management separated from the backend processing. The installed example is an application of this kind.

The starting point is a character based screen whose Screen Section definition is as follows:

```
       01  s1.
           03 "Customer code:"        line 3 col 2.
           03 using cust-code         col + 2 high prompt.
           03 "First name: "          line 4 col 2.
           03 using Cust-First-Name   col + 2 high prompt.
           03 "Last name:"            line 5 col 2.
           03 using Cust-Last-Name    col + 2 high prompt.
           03 "Address:"              line 7 col 2.
           03 "Street:"               line 8 col 2.
           03 using Cust-Street       col + 2 high prompt.
           03 "City:"                 col 50.
           03 using Cust-City         col + 2 high prompt.
           03 " State:"               line 9 col 2.
           03 using Cust-State        col + 2 high prompt.
           03 "Zip code:"             col 50.
           03 using Cust-Zip          col + 2 high prompt.
           03 "Gender:"               line 11 col 2.
           03 using Cust-Gender       col + 2 high prompt.
           03 "Phone number:     "    line 13 col 2.
           03 using Cust-Phone        col + 2 high prompt.
           03 "Cell Phone number:"    line 14 col 2.
           03 using Cust-CellPhone    col + 2 high prompt.

       01  s-func.
           03 "F1=Lookup"             line 23 col 2 reverse.
           03 "F3=Delete"             col + 2 reverse.
           03 "F5=First"              col + 2 reverse.
           03 "F6=Prev"               col + 2 reverse.
           03 "F7=Next"               col + 2 reverse.
           03 "F8=Last"               col + 2 reverse.
           03 "F9=Save"               col + 2 reverse.
           03 "ESC=Exit"              col + 2 reverse.
```

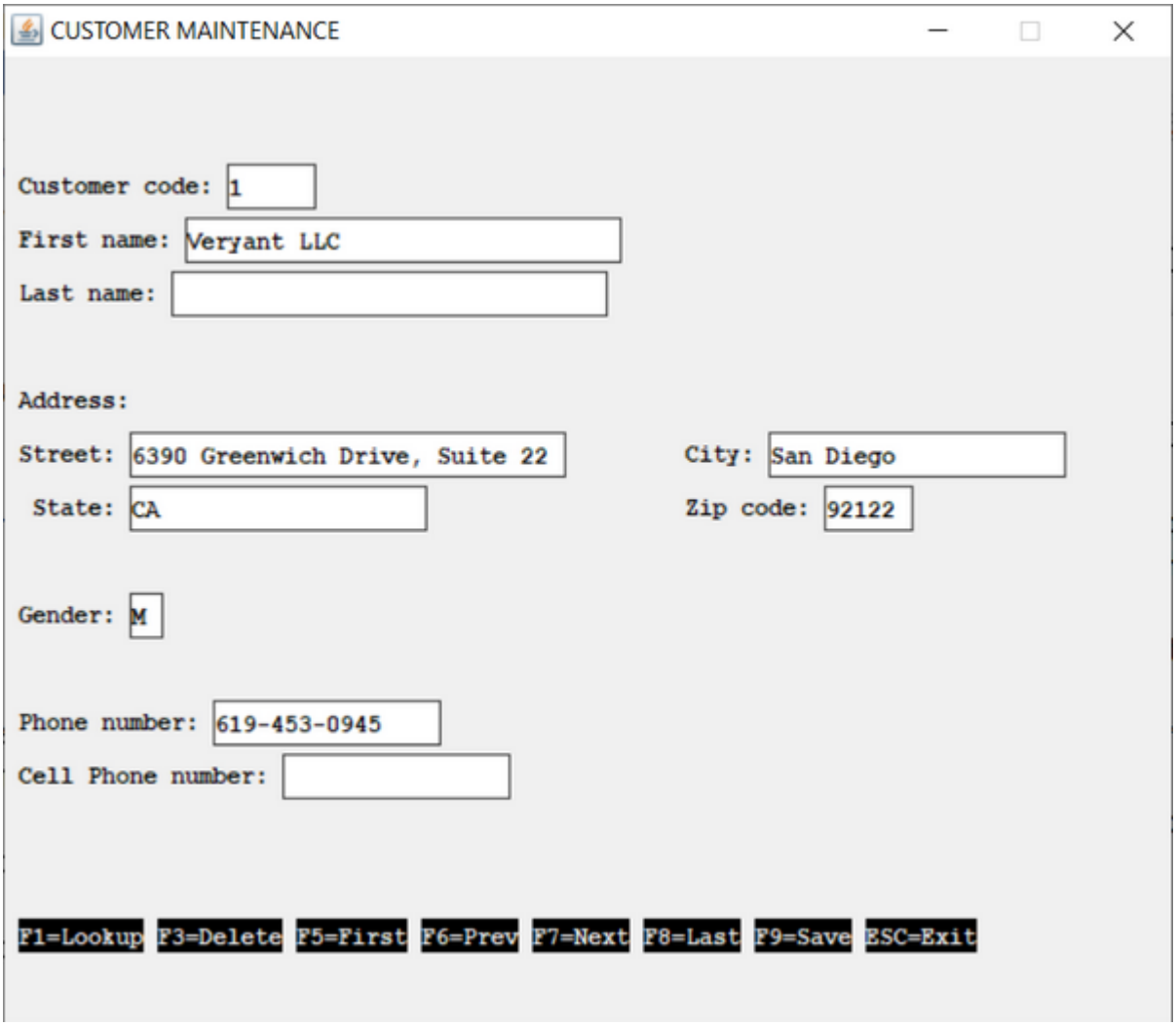
At runtime, the screen appears as follows:


CUSTOMER
CUSTOMER MAINTENANCE
Customer code: 1
First name: Veryant LLC
Last name:
Address:
Street: 6390 Greenwich Drive, Suite 22
City: San Diego
State: CA
Zip code: 92122
Gender: M
Phone number: 619-453-0945
Cell Phone number:
F1=Lookup F3=Delete F5=First F6=Prev F7=Next F8=Last F9=Save ESC=Exit

The minimal modification that can be done to the Screen Section in order to obtain a graphical screen is to change text labels and FROM fields to LABEL controls and USING fields to ENTRY-FIELD controls as follows:

```
01  S1.
    03 label "Customer code:"             line 3 col 2.
    03 entry-field using cust-code        col + 2 high prompt.
    03 label "First name: "               line 4 col 2.
    03 entry-field using Cust-First-Name  col + 2 high prompt.
    03 label "Last name:"                 line 5 col 2.
    03 entry-field using Cust-last-Name   col + 2 high prompt.
    03 label "Address:"                   line 7 col 2.
    03 label "Street:"                    line 8 col 2.
    03 entry-field using Cust-Street      col + 2 high prompt.
    03 label "City:"                      col 50.
    03 entry-field using Cust-City        col + 2 high prompt.
    03 label " State:"                    line 9 col 2.
    03 entry-field using Cust-State       col + 2.
    03 label "Zip code:"                  col 50.
    03 entry-field using Cust-Zip         col + 2 high prompt.
    03 label "Gender:"                    line 11 col 2.
    03 entry-field using Cust-Gender      col + 2 high prompt.
    03 label "Phone number:"              line 13 col 2.
    03 entry-field using Cust-Phone       col + 2 high prompt.
    03 label "Cell Phone number:"         line 14 col 2.
    03 entry-field using Cust-CellPhone   col + 2 high prompt.

01  s-func.
    03 label "F1=Lookup"             line 17 col 2 reverse.
    03 label "F3=Delete"             col + 2  reverse.
    03 label "F5=First"              col + 2 reverse.
    03 label "F6=Prev"               col + 2 reverse.
    03 label "F7=Next"               col + 2 reverse.
    03 label "F8=Last"               col + 2 reverse.
    03 label "F9=Save"               col + 2 reverse.
    03 label "ESC=Exit"              col + 2 reverse.
```

After this modification, the screen appears as follows:

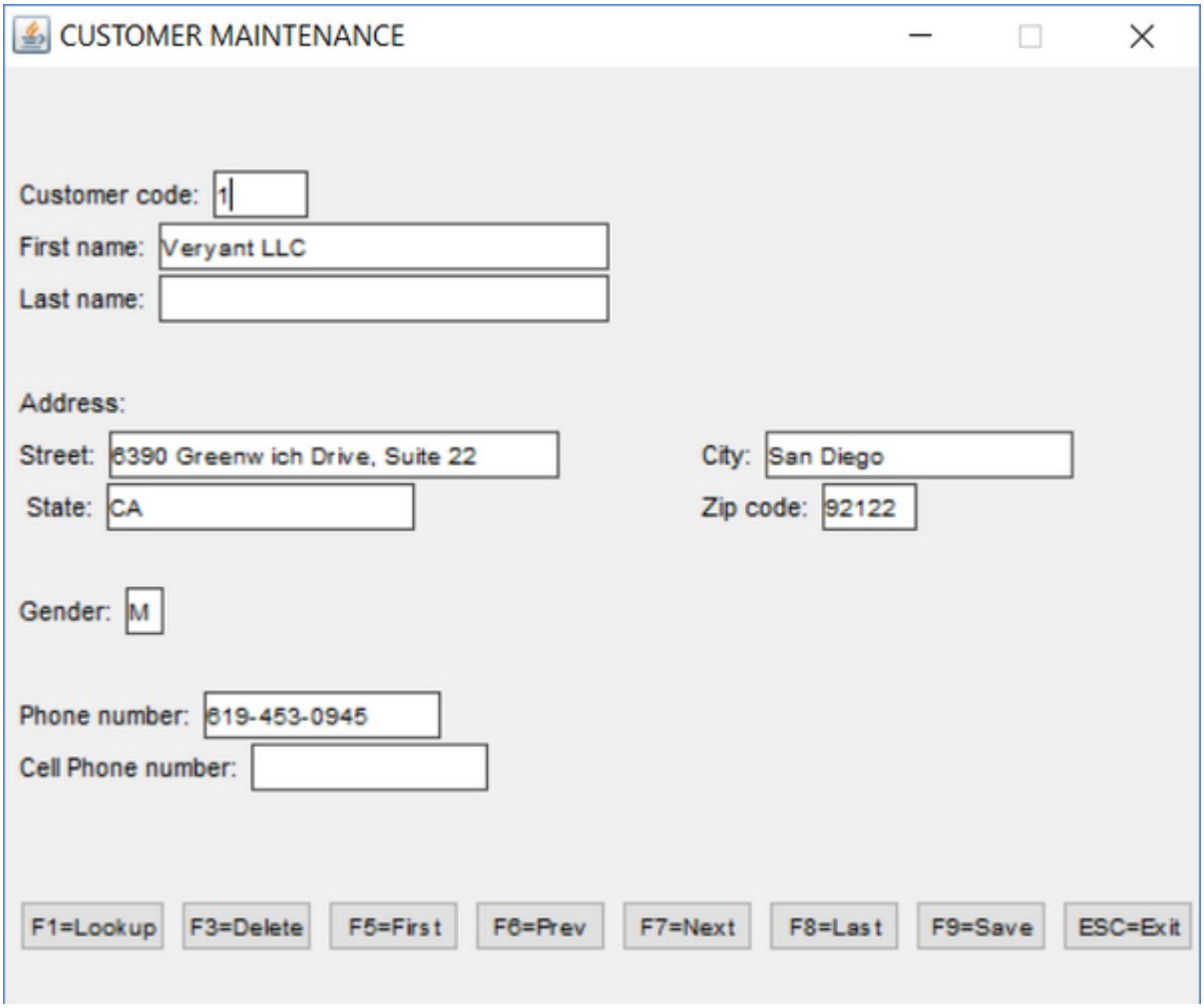


A little more complex modification is to add some properties and styles to LABEL and ENTRY-FIELD controls and replace text labels that describe an action with graphical PUSH-BUTTON controls as follows:

```
       01  S1.
           03 label
              title "Customer code:"     line 3 col 2.
           03 entry-field
              numeric
              value cust-code            col + 2.
           03 label
              title "First name: "       line 4 col 2.
           03 entry-field
              value Cust-First-Name      col + 2.
           03 label
              title "Last name:"         line 5 col 2.
           03 entry-field
              value Cust-last-Name       col + 2.
           03 label
              title "Address:"           line 7 col 2.
           03 label
              title "Street:"            line 8 col 2.
           03 entry-field
              value Cust-Street          col + 2.
           03 label
              title "City:"              col 50.
           03 entry-field
              value Cust-City            col + 2.
           03 label
              title " State:"            line 9 col 2.
           03 entry-field
              value Cust-State           col + 2.
           03 label
              title "Zip code:"          col 50.
           03 entry-field
              value Cust-Zip             col + 2.
           03 label
              title "Gender:"            line 11 col 2.
           03 entry-field
              value Cust-Gender          col + 2.
           03 label
              title "Phone number:"      line 13 col 2.
           03 entry-field
              value Cust-Phone           col + 2.
           03 label
              title "Cell Phone number:" line 14 col 2.
           03 entry-field
              value Cust-CellPhone       col + 2.

       01  s-func.
           03 push-button
              exception-value 1
              self-act
              title "F1=Lookup"          line 17 col 2
              size 9.
```

```
           03 push-button
              exception-value 3
              self-act
              title "F3=Delete"          col + 2.
           03 push-button
              exception-value 5
              self-act
              title "F5=First"           col + 2.
           03 push-button
              exception-value 6
              self-act
              title "F6=Prev"            col + 2.
           03 push-button
              exception-value 7
              self-act
              title "F7=Next"            col + 2.
           03 push-button
              exception-value 8
              self-act
              title "F8=Last"            col + 2.
           03 push-button
              exception-value 9
              self-act
              title "F9=Save"            col + 2.
           03 push-button
              exception-value 27
              self-act
              title "ESC=Exit"           col + 2.
```

After this modification, the screen appears as follows:



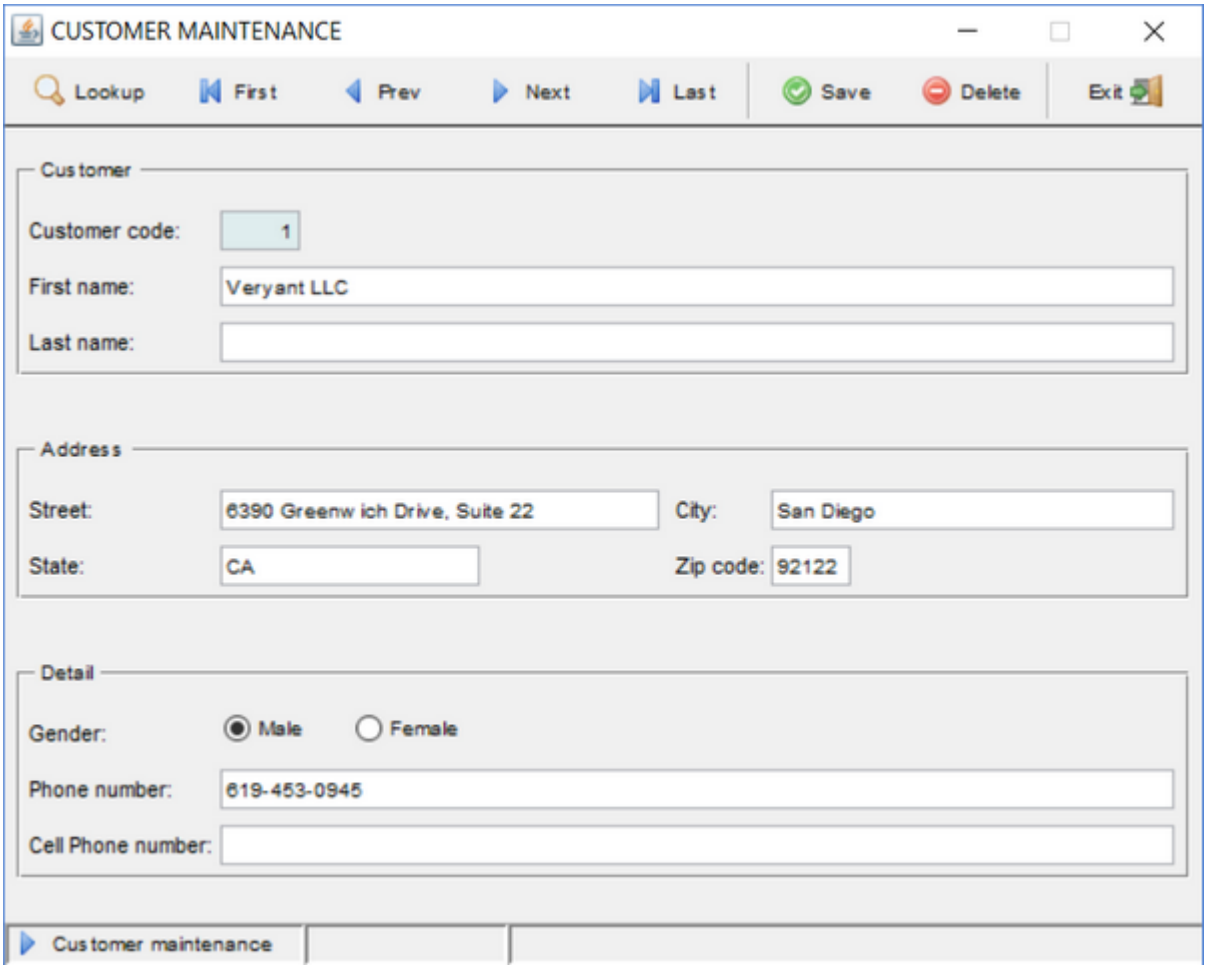
Note - if you wish to assign a special attribute to all the fields, for example if you want all the push-buttons to have the SELF-ACT style, you can use the compiler configuration property iscobol.compiler.gui.<control-type>.defaults instead of writing "SELF-ACT" in every push-button description in the Screen Section.

Though the screen is graphical it's still not perfect; it doesn't look like a modern and appealing screen. A further effort is required in order to obtain a good screen. The following changes are suggested:

- Reorder the graphical control and put frames around them to better describe the information that they host,
- Add icons to the buttons and move them to a tool-bar on top of the screen
- Use different controls where applicable. For example the Gender can be accepted using two radio-buttons instead of an entry-field. This will save you for checking if the user input a valid value.
- Add a status-bar

Refer to CUSTOMER.cbl in *sample/modernization/graphical/3-full-gui* for the necessary code.

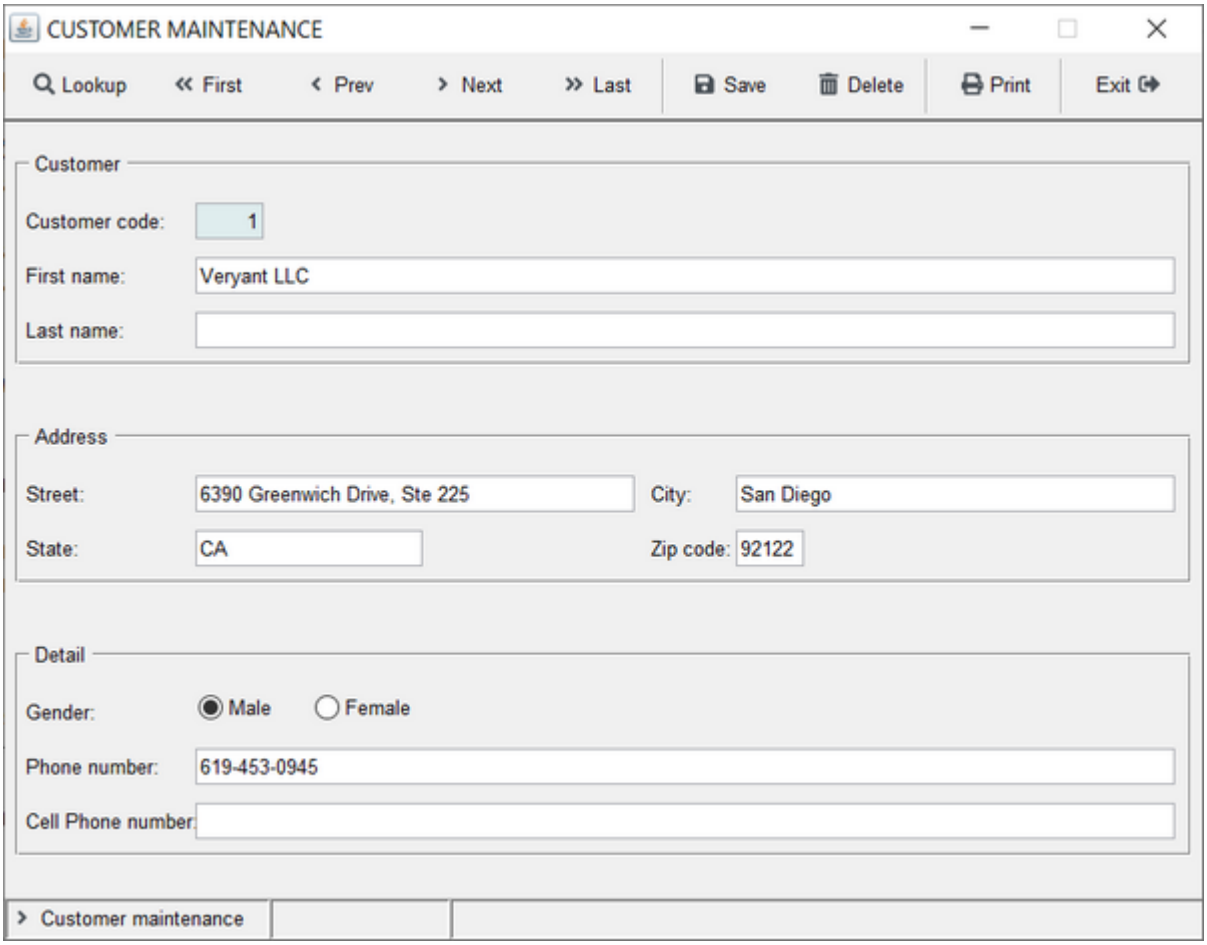
After these changes, the screen will look like this:



Button icons could be replaced with glyphs retrieved from a symbol font, like Font Awesome.

Refer to CUSTOMER.cbl in *sample/modernization/graphical/4-advanced-gui* for the necessary code.

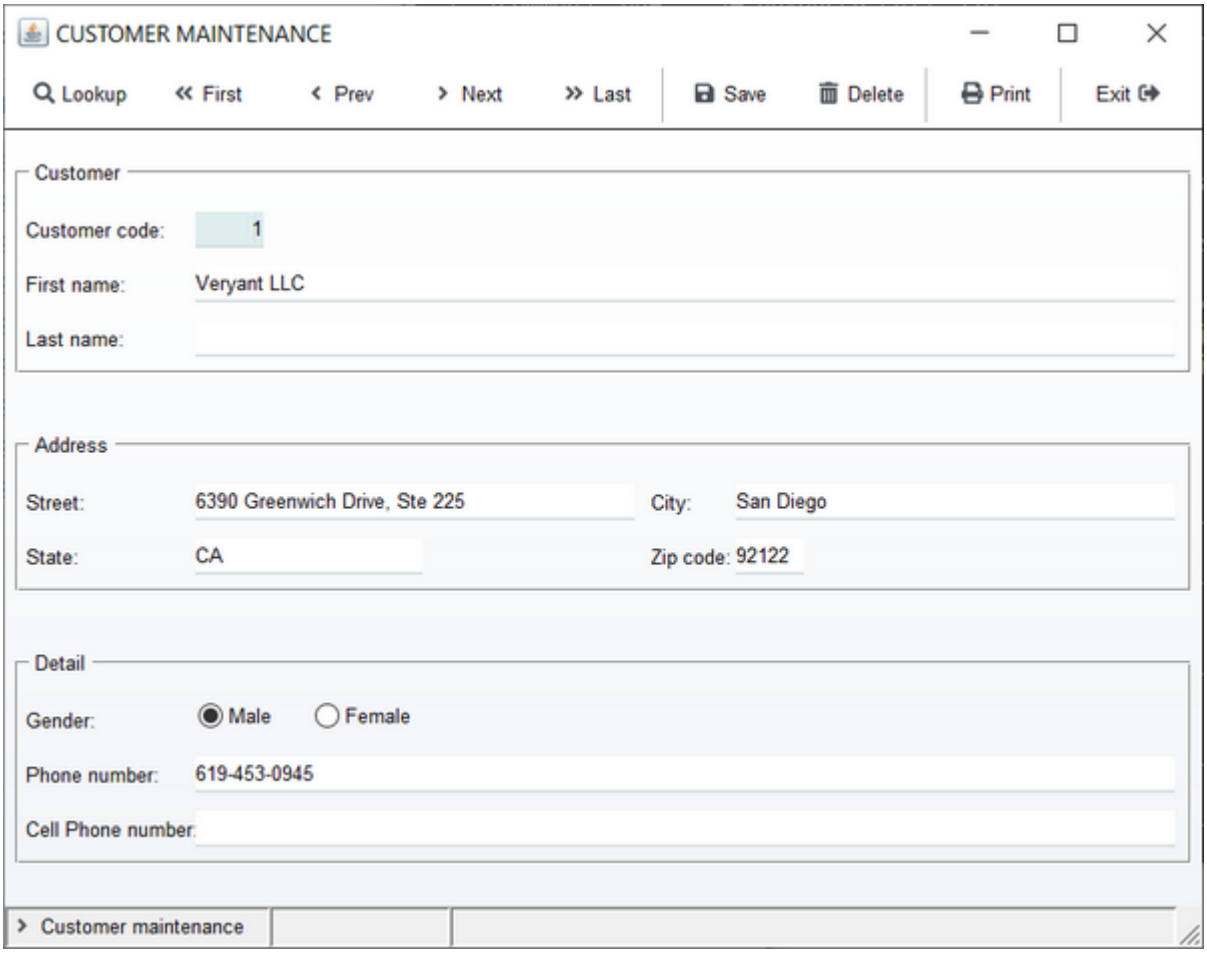
After this change, the screen will look like this:


CUSTOMER MAINTENANCE
Lookup
First
Prev
Next
Last
Save
Delete
Print
Exit
Customer
Customer code:
1
First name:
Veryant LLC
Last name:
Address
Street:
6390 Greenwich Drive, Ste 225
City:
San Diego
State:
CA
Zip code:
92122
Detail
Gender:
Male
Female
Phone number:
619-453-0945
Cell Phone number:
Customer maintenance

An additional cosmetic improvement can be introduced at compile time by setting iscobol.compiler.gui.<control-type>.defaults to apply properties and styles to all the controls without code changes. Let's consider, for example, the following Compiler configuration:

```
# Compiler.regexp to remove the 3D and "ERASE" styles when displaying the window
iscobol.compiler.regexp="(?i)( 3-D,)" "" \
                        "(?i)( 3-D)" "" \
                        "(?i)( ERASE,)" "" \
                        "(?i)( ERASE)" ""

#### code injection for controls ####
# add the gradient color on all windows
iscobol.compiler.gui.window.defaults= \
    gradient-color-1 rgb x#FFFFFF \
    gradient-color-2 rgb x#F2F5F9  \
    gradient-orientation gradient-northeast-to-southwest

# add the transparent style to all labels, check-boxes and radio-buttons
iscobol.compiler.gui.label.defaults= transparent
iscobol.compiler.gui.check_box.defaults= transparent
iscobol.compiler.gui.radio_button.defaults= transparent
iscobol.compiler.gui.frame.defaults= transparent

# set the white color for all toolbars
iscobol.compiler.gui.tool_bar.defaults= background-Color rgb x#FFFFFF \
                                        foreground-Color rgb x#000000

# add the flat style to all push buttons
iscobol.compiler.gui.push_button.defaults=flat

# add the underline style to all entry-fields
iscobol.compiler.gui.entry_field.defaults = border-width (0, 0, 2, 0) \
                                            border-color rgb x#DAE1E5 \
                                            margin-width (3, 3, 3, 3)
```

After recompiling the program with the above settings, the screen will look like this:

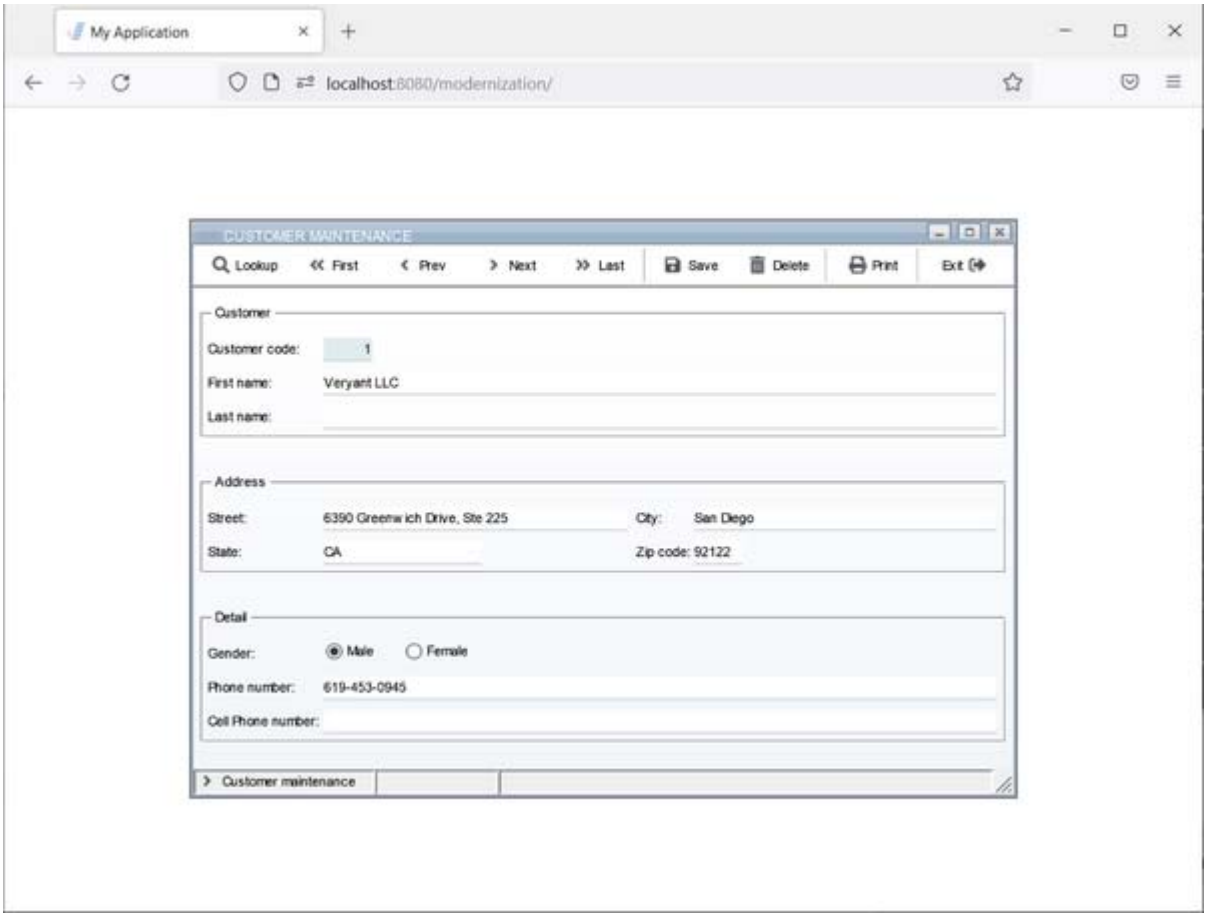


The last step in the modernization process is to bring our user interface to the web.

It can be done in three ways, discussed below:

WebClient

WebClient allows you to display your screen in a web browser. The screen that the user sees in the browser is identical to the screen he would see by running the desktop application.

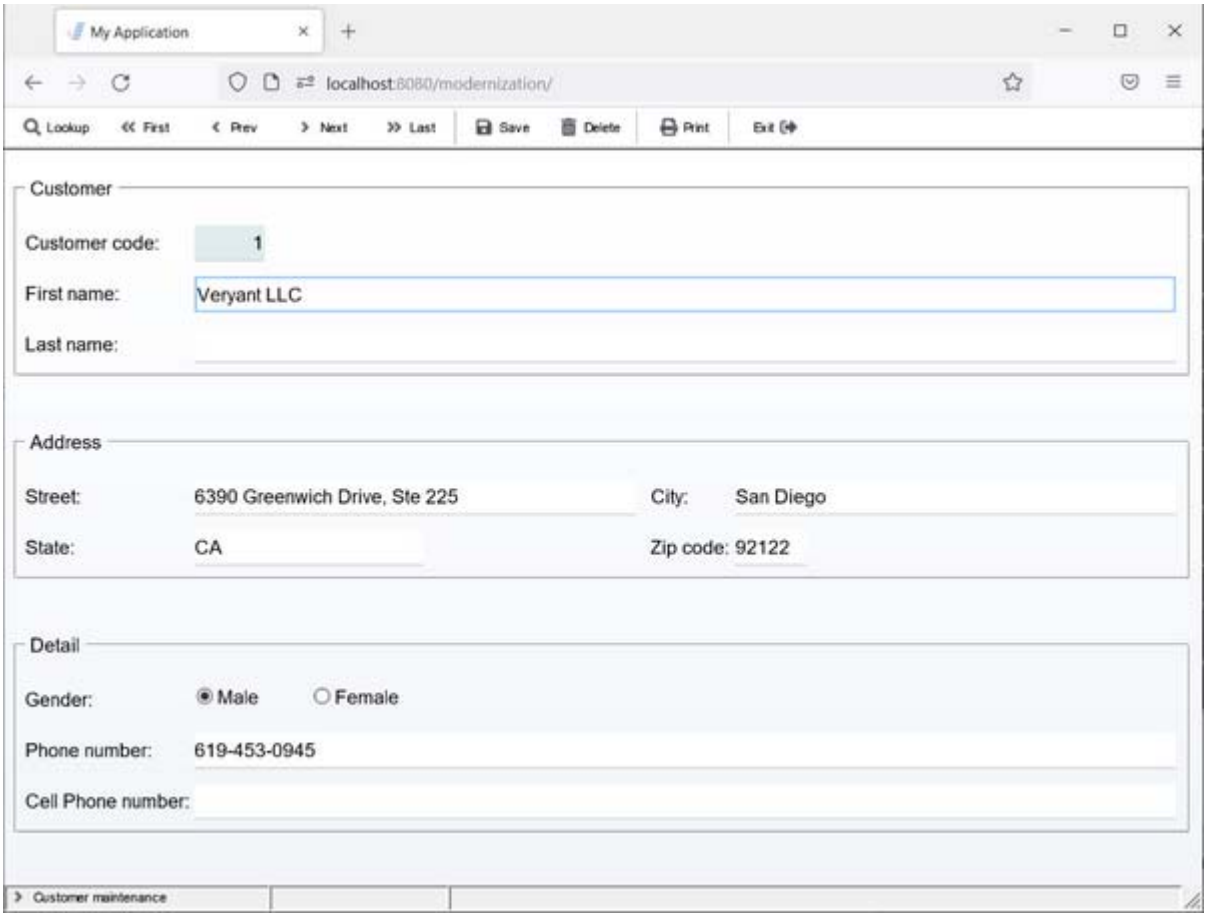


With this solution no change is required to the user interface management code.

However, by default the program doesn't look like a web application, it looks more like a remote desktop implemented in the browser.

In order to make the program more similar to a web application, you should undecorate and maximize the window as demonstrated in *sample/modernization/graphical/4-advanced-gui*.

After this small change, the screen will look like this:

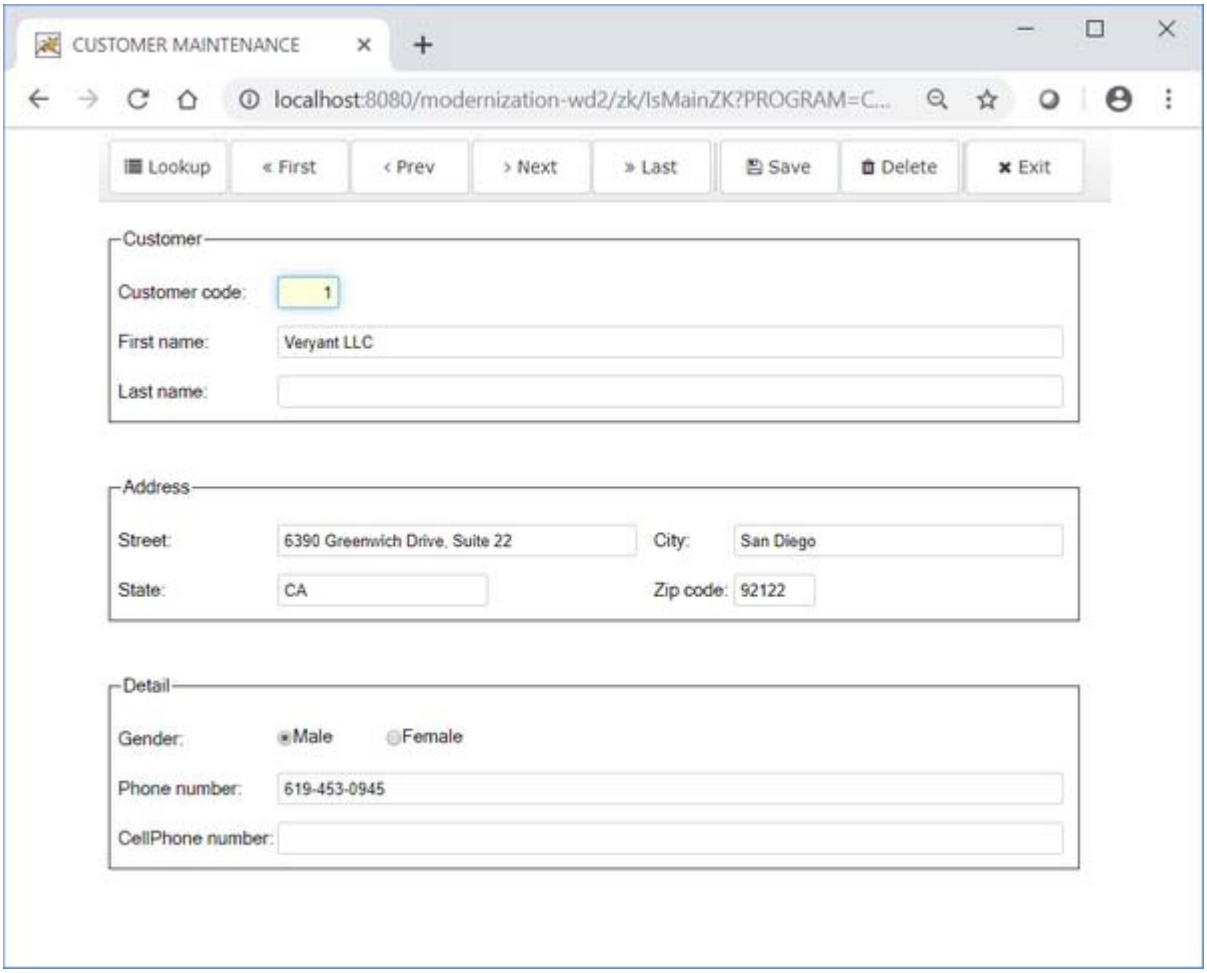


For more information about WebClient, see isCOBOL Evolve: WebClient.

A complete and working example is provided in the folder *sample/modernization/webclient*.

WebDirect

The WebDirect option allows you to display your screen in a web browser. The screen is rendered using the ZK Framework so the layout is a little different than the one you would see by running the desktop application.



Unlike WebClient, WebDirect doesn't provide a full compatibility with the Swing components used by the desktop application. In particular:

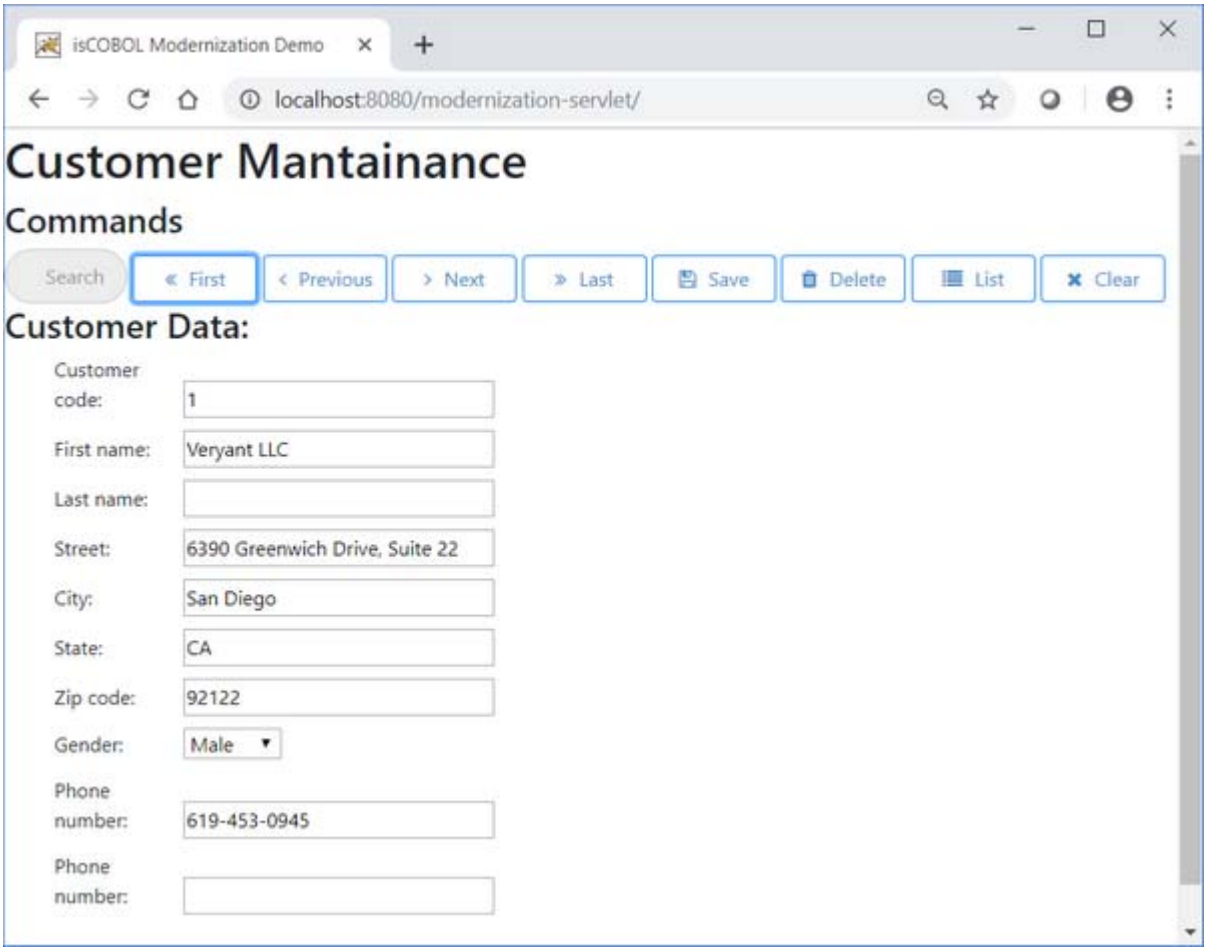
- some control properties and styles are not supported,
- java-beans based on Swing and JavaFX components must be replaced with the proper ZK component.

For more information about the Web Direct 2.0 option, see WebDirect option.

A complete and working example is provided in the folder *sample/modernization/eis/wd2*.

Servlet option

This kind of option allows you to obtain a real web application. The user interface management must be rewritten in HTML5, CSS and JavaScript. The COBOL code is maintained only for the backend processing.



For more information about servlets written in COBOL, see COBOL Servlet option (OOP).

A complete and working example is provided in the folder *sample/modernization/eis/servlet*.

Printing modernization (from text-only to graphical, from paper to PDF)

Most legacy COBOL programs wrote text-only printouts using a monospaced font so that the text can be aligned and distributed into columns by inserting space characters among words. Basic graphics like bars and boxes are obtained by printing pipes, dashes or underscores.

isCOBOL allows you to modernize these printouts through the WIN$PRINTER library routine. With this routine you can

- use different true type fonts, with different styles and colors,
- distribute text into columns even if the font is not monospaced,
- draw graphic boxes with square or round corners,
- print image pictures.

In addition, the isCOBOL runtime allows you to redirect the print job to three possible destinations:

- The system spooler

When the print job is sent to the system spooler, it causes the active printer to manage it. The active printer can be changed through the WIN$PRINTER routine.

- The print preview

  When the print job is sent to the print previewer, the runtime shows a print preview dialog for the user to review the printout. From this dialog the user can either print to a phyisical printer or save to a PDF file.

- PDF file

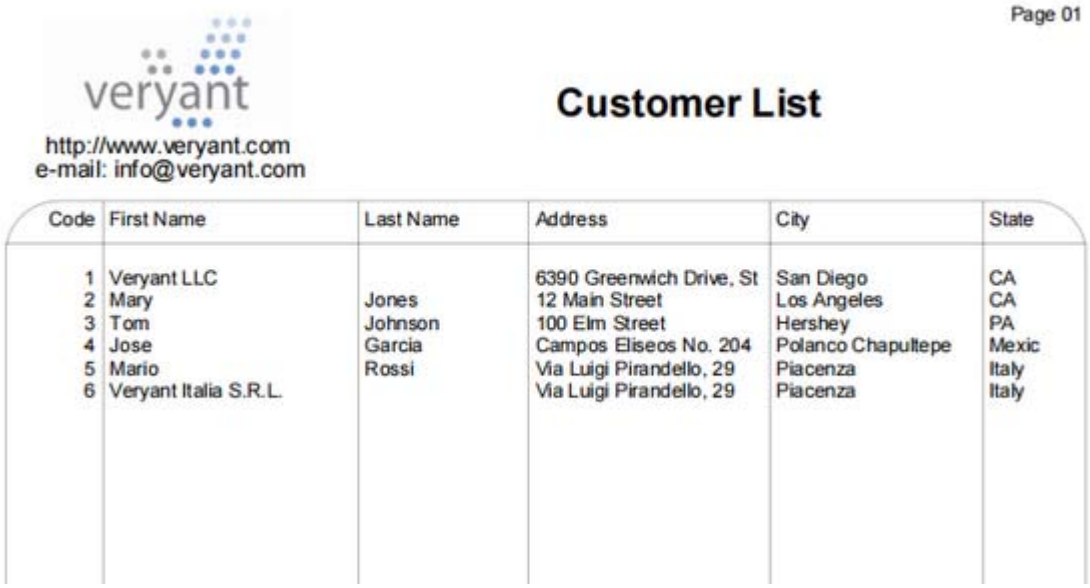
  When the print job is sent to a PDF file, the runtime takes care of generating a PDF file on disc.

The destination is controlled by the physical name of the print file, and no specific coding is required in the print procedure logic.

The modernization sample installed along with isCOBOL under *sample/modernization* includes a print program named *PRINTCUSTOMER.cbl* that evolves along with the UI, from a text-only printout

```
Customer List                                                          Page  1
================================================================================
Code First Name      Last Name       Address                  City
================================================================================
   2 Mary            Jones           12 Main Street           Los Angeles
   3 Tom             Johnson         100 Elm Street           Hershey
   4 Jose            Garcia          Campos Eliseos No. 204   Polanco Chapultepe
   5 Mario           Rossi           Via Luigi Pirandello, 29 Piacenza
   6 Veryant Italia                  Via Luigi Pirandello, 29 Piacenza
```

to a better graphical printout



Review the code of *PRINTCUSTOMER.cbl* in its various versions to understand the changes that are necessary for this kind of evolution.