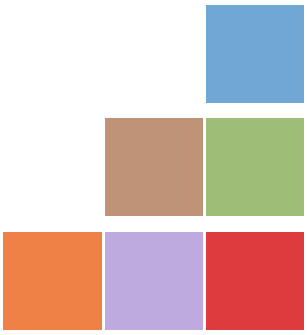
The CitSQL® Family of Precompilers

Getting Started
Version 1.3



Contents

Acknowledgment

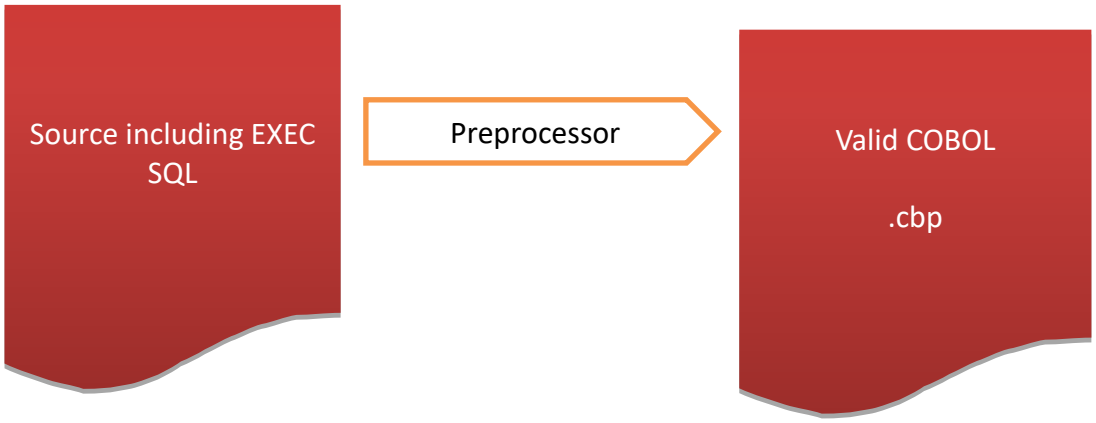
Contact Information:

COBOL-IT
The Lawn
22-30 Old Bath Road
Newbury, Berkshire, RG14 1QN
United Kingdom
Tel: +44-0-1635-565-200

Introduction

Overview

The CitSQL Family of Products allow COBOL source code that contains EXEC SQL statements to be pre-processed, with the resulting output file being a COBOL source file, in which the EXEC SQL statements have been commented out, and replaced with CALL statements. This file, which is generated with a .cbp extension by default, can then be compiled by your COBOL-IT Compiler, and the resulting object code can be run to provide connectivity with one of the CitSQL-supported databases.



The CitSQL Family of Products

CitSQL is available as separate products for the MySQL®, PostgreSQL®, and for relational databases which have an ODBC interface, such as SQL Server®. CitSQL, provides an SQL precompiler for COBOL, and a runtime component for all of its products. CitSQL provides an SQL Engine Client Component for CitSQL for MySQL, and CitSQL for PostgreSQL. Note that while CitSQL for ODBC does not distribute the odbc client component, this component is available on all Windows platforms that support 32-bit applications.

Product	Precompiler	Runtime Component	SQL Engine Client Component
CitSQL for MySQL (Windows)	bin\citmysql.exe	bin\rcqmysql.dll	bin\libmysql.dll
CitSQL for MySQL (Linux/Unix)	bin/citmysql	bin/rcqmysql.so	lib/libmysql.so
CitSQL for PostgreSQL (Windows)	bin\citpgsql.exe	bin\rcqpgsql.dll	bin\libpq.dll
CitSQL for PostgreSQL (Linux/Unix)	bin/citpgsql.so	bin/rcqpgsql.so	lib/libpq.so
CitSQL for ODBC	bin\citodbc.exe	bin\rcqodbc.dll	odbc32.dll**

**odbc32.dll is not distributed with CitSQL for ODBC.

When using CitSQL, the normal sequence of events is :

- Step 1-Precompile source files containing ESQL COBOL statements
  - For Step 1, you will need your precompiler to be in your PATH. The precompiler will need to locate a file called SQLCA, which is located in the include subdirectory of the root installation directory. A typical precompile command line is :

    >citmysql :IncludeSearchPath=./include myprog.cob

    Where the program being recompiled has been located in the parent directory of the include directory containing the file SQLCA.

  - The default behavior of of the precompiler is to create a target file called myprog.cbp, in which SQL statements have been commented out, and replaced with COBOL CALL statements to the CitSQL runtime component.

- Step 2- Compile the target source files creating executable object files
  - For Step 2, you will need to have your COBOL-IT Compiler Suite Enterprise Edition installed. The COBOL-IT Compiler Suite Enterprise Edition includes a setup file, called setenv_cobolit.bat in Windows environments, and called cobol-it-setup.sh in Linux/Unix environments. The Compiler setup file must be run prior to using the COBOL-IT compiler..
    A typical command line for using the COBOL-IT compiler is:

    >cobc myprog.cbp

  - The compiler will then create a shared object- myprog.dll in a Windows environment, or myprog.so in a Linux/Unix environment.

- Step 3- Execute the compiled object files
  - For Step 3, the database server must be running. You will need to have the COBOL-IT runtime in your path. This is done when you run the setup file for the compiler, Then, you must have the compiled objects, the runtime component of CitSQL, and the shared library/dll through which the runtime component communicates with the database locatable by the COB_LIBRARY_PATH environment variable.

    The following files should then be located, either in a directory that is in the PATH, or in a directory that is in the COB_LIBRARY_PATH. This can be accomplished by explicitly setting these environment variables to the locations of these files, or by moving these files into directories that are in directories that are in your PATH or COB_LIBRARY_PATH.

    Consider an example using CitSQL for MySQL:

COBOL-IT Runtime	Cobcrun
CitSQL Runtime component	rcqmysql.dll / rcqmysql.so
SQL Engine Client component	libmysql.dll

Compiled Object myprog.dll / myprog.so

On Linux/Unix platforms, the SQL Engine Client component should then be located in the COBOL-IT installation lib directory, so that it will automatically be in the LD_LIBRARY_PATH, after the cobol-it-setup.sh script has been run.

SQL Engine Client component libmysql.so

Example:
Note- In these examples, the files in the PreprocessorDistrib\bin directory are copied into the COBOLITDIR\bin directory. In many cases, this will not be the preferred solution, in which case you should explicitly set the PATH environment variable to include the PreprocessorDistrib\bin directory.

In Windows:
>setenv_cobolit.bat
>copy PreprocessorDistrib\bin*.* %COBOLITDIR%\bin
>set COB_LIBRARY_PATH=[location of myprog.dll]
>cobcrun myprog
In Unix: (from the PreprocessorDistrib directory)
>source /opt/cobol-it/bin/cobol-it-setup.sh
>cp bin/* $COBOLITDIR/bin
>cp lib* $COBOLITDIR/lib
>export COB_LIBRARY_PATH=[location of myprog.so]
>cobcrun myprog

- The program that is being run contains CALLs to the CitSQL runtime component which in turn calls the SQL Engine Client component, which performs database operations, and returns information to the runtime component.

The CitSQL Precompiler

The CitSQL precompiler parses COBOL source programs containing SQL statements, and generates new COBOL target programs, in which the embedded SQL statements have been replaced with COBOL CALL statements that reference the CitSQL runtime component. The new COBOL program is created with a .cbp extension.
The CitSQL Precompiler is used at Development time only, and is used to transform programs containing ESQL COBOL statements prior to compiling with the COBOL-IT Compiler Suite Enterprise Edition.

The CitSQL Runtime Component

The CitSQL runtime component, which is the target of the COBOL CALL statements generated by the CitSQL precompiler, manages the connection with the database through the Client Component of the Database SQL Engine.

The CitSQL Runtime Component is used at Runtime only, and must be distributed with your compiled application.

The SQL Engine Client Component

The SQL Engine Client Component for MySQL and PostgreSQL is distributed with CitSQL. The SQL Engine Client component sits between the CitSQL Runtime Component, and the Database.

The SQL Engine Client Component is used at Runtime only, and must be distributed with your compiled application. Note, however, that the end-user is responsible for acquiring the necessary licenses for using the SQL Engine Client Component from the database vendor.

What differentiates CitSQL

In many ways, CitSQL is similar to existing SQL precompilers on the market. While it is intended to be used as a replacement for existing product, there are important differences. Most notably:

- The SQL DECLARE clause is accepted but ignored. The entire DATA DIVISION is analyzed, and all the variables can be referred to in SQL statements, wether they are within an SQL DECLARE block or not.
- Each CitSQL product is optimized for its target. When targeting MySQL and PostgreSQL, CitSQL produces precompiled source programs that target the database using native API's, optimized for the target database.
- When targeting ODBC, CitSQL produces precompiled source programs optimized for the ODBC standard.
- CitSQL recognizes a superset of many existing SQL dialects, and in some cases will adapt the SQL statements for the target database in question (see Section 4).
- The precompiled source code produced by CitSQL produces calls to the runtime component, which is included with the product as either a DLL (Windows) or shared object (Linux/Unix). There is no need to statically link the runtime component to the program.

- If CitSQL fails to recognized an SQL statement, and detects no reference to a host variable, it will transmit the SQL statement "as-is" to the database engine. (see Section 5.1)

Getting Started

Installation

CitSQL is distributed as a tar.gz archive on Unix and a Zipped (.zip) archive on Windows.

Product	**Distributed Archive**
CitSQL for MySQL	distribMYSQL.tar.gz
CitSQL for PostgreSQL	distribPGSQL.tar.gz.
CitSQL for ODBC	distribODBC.zip

To install CitSQL, you must :

- Copy the zipped archive into the COBOL-IT installation directory. See table below for defaults for different platforms:

Platform	COBOL-IT Install Directory	COBOL-IT setup script
32-bit Linux/Unix	/opt/cobol-it	/opt/cobol-it/bin/cobol-it-setup.sh
64-bit Linux/Unix	/opt/cobol-it-64	/opt/cobol-it-64/bin/cobol-it-setup.sh
32-bit Windows	C:\COBOL\COBOLIT	C:\COBOL\COBOLIT\setenv_cobolit.bat
64-bit Windows	C:\COBOL\COBOLIT64	C:\COBOL\COBOLIT64\setenv_cobolit.bat

- Unzip/Untar the archive, creating the subdirectory distribXXSQL (where XX is MY, PG or ODBC).
- Set the PATH environment variable to include the PreprocessorDistrib\bin directory.
- (Linux/UNIX) Set the LD_LIBRARY_PATH to include the PreprocessorDistrib\lib directory.
- Run the COBOL-IT setup script.

Directory Structure

The process of unzipping the archive will create a directory called distribXXSQL, which contains the following directory structure:

Directory	Contains
/bin	The Precompiler and Runtime Component. (Windows): The Client Component of the Database Engine.
/include	The SQLCA COBOL structure
/lib	(Linux/UNIX): The Client Component of the Database Engine
/samples/cobol	A sample COBOL application that can be used to test that your installation of CitSQL.
/doc	Documentation

Required Third-Party Tools

Product	Required Tools
MySQL	Client Tools, version 5 or above
PostgreSQL	Client Tools, version 9 or above
ODBC	ODBC Library

As an example:
For a default installation of the PostgreSQL Development tools :
On Linux Redhat, for PostgreSQL :

>yum install postgresql-devel

Using the Precompiler

Note- for the purposes of this documentation, the command-line notation of CitSQL is intended to refer to the CitSQL precompiler. Depending on which version of CitSQL is being used, the actual name of the precompiler executable will differ.

CitSQL for MySQL	citmysql
CitSQL for PostgreSQL	citpgsql
CitSQL for ODBC	citodbc

License

Usage of the CitSQL Precompiler requires a valid license.

COBOL-IT has made changes to licensing to all products, effective with the the release of COBOL-

IT Compiler Suite version 4.0 (and later). With the release of COBOL-IT Compiler Suite version 4.0, separate license files are required for each product/platform pairing. As a consequence, users can no longer use the same license for multiple products, on multiple platforms; users deploying multiple products require multiple license files.

Default location

The default location for COBOL-IT product license files in %COBOLITDIR% (Windows) and $DEFAULT_CITDIR (Linux). For COBOLIT Compiler Suite version 4, the default installation directory is /opt/cobol-it4-64 (Linux).

Default naming convention

License files located in the default location named “citlicense.xml”, or with names prefixed by “citlicense“ and with the “.xml” extension will be tested for validation by COBOL-IT products. As examples, COBOL-IT products would automatically test licenses named citlicense.xml, citlicense-001.xml, citlicense-myproduct.xml.

Using COBOLIT_LICENSE to reference single or multiple license files

For cases where different naming conventions are used, or where license files are not stored in the default installation directory, the user should use the COBOLIT_LICENSE environment variable to indicate the full path(es) and name(s) of their license file(s).

When a Subscription is registered, the registered user receives download authorization to the CitSQL Precompiler for the appropriate database, and a license file called citlicense.xml, which is generated to match the duration of the registered subscription.

Note that CitSQL Precompiler Subscriptions and licenses are defined as lasting for a prescribed period of time, from the date of the generation of the Subscription and corresponding license. That is, a one-year Subscription is accompanied by a one-year license, and the expiration date is set at one-year after the generation of the license, -not- one year after the installation of the software.

The CitSQL Precompiler and CitSQL Runtime Component search for a license file in the following manner:

1- Check to see if the COBOLIT_LICENSE environment variable is set.
COBOLIT_LICENSE, if set, describes the full path, and license name to be used by the COBOL-IT Compiler, Compler-generated executables, and Runtime.

Example:
For Linux/Unix-based platforms:
>export COBOLIT_LICENSE=/opt/cobol-it/license/mycitlicense.xml

Note that when indicating multiple license files, the semicolon “ ; ” separator is used. In Linux, the list of license files is started and finished with single-quote marks “ ‘ “. The single-quote is located on the same key as the double-quote on most keyboards.

Example:

>export COBOLIT_LICENSE='/opt/cobol-it4-64/compilerlic.xml;/opt/cobol-it4-64/citsqllic.xml'

For Windows-based platforms:
>set COBOLIT_LICENSE=C:\COBOL\COBOLIT\license\mycitlicense.xml
>set COBOLIT_LICENSE=C:\COBOL\COBOLIT\mylic.xml;C:\COBOL\COBOLIT\mysqllic.xml

2- Check to see if the COBOLITDIR environment variable is set. If set, check for a file called citlicense.xml located in the directory defined by the COBOLITDIR environment variable.

Example:
For Linux/Unix-based platforms:
>export COBOLITDIR=/opt/cobol-it

For Windows-based platforms:
>set COBOLITDIR=C:\COBOL\COBOLIT

3- If neither the COBOLIT_LICENSE or COBOLITDIR environment variables are set, check for a file called citlicense.xml located in the default installation directory. On Linux/Unix platforms, the default installation directory is /opt/cobol-it. On Windows platforms, the default installation directory is C:\COBOL\COBOLIT for 32-bit product, and C:\COBOL\COBOLIT64 for 64-bit product.

For information about your license, including version, owner, and expiration date of the license, type:

For CitSQL for MySQL	***>citmysql***
For CitSQL fro PostgreSQL	***>citpgsql***
For CitSQL for ODBC	***>citodbc***

Usage

>citsql [options] [source file(s)] or

>citsql @stdoptions.res @stdprogs.res

Options

CitSQL Options include:

CitSQL **Command-Line Option**	**Description**
:CloseOnCommit=[True/False]	Aligns CitSQL behavior with Oracle behavior in cases when a BEGIN … END construct contains an EXEC SQL SELECT statement followed by a COMMIT statement. When :CloseOnCommit=True, the preprocessor does not close the SQL cursor when the SELECT statement has executed. Instead, the cursor is closed after the COMMIT statement is executed.
:CursorSynteticName=[True/False]	Causes the Cursor Name to be generated dynamically at runtime by the RCQ runtime.
:DBEncoding= <Codepage or UTF-8>	Specifies what encoding is used by the DB storage.
:DeallocateCloseCursor=[True/False]	Causes the CLOSE CURSOR statement to also deallocate the DECLAREd cursor.
:DefaultCCSID=<Codepage or UTF-8>	Specifies the default CCSID for string fields that have no explicit CCSID declaration.
:DebugMode= [TRUE/FALSE]	Default [FALSE] When set to TRUE, causes log file created when LogMode=TRUE to contain more detail in some situations.
ExpandIncludes= [TRUE/FALSE]	Default [FALSE] When set to TRUE, causes all COPY files in the program to be expanded into the source file prior to pre-compiling the source. Note- If any SQL statements reference a field in a COPY book you must use the command-line option: :ExpandIncludes=True
ForceStringMode=[TRUE/FALSE]	Default is [FALSE] unless :DBEncoding is set to UTF-8 or utf8. Then, the default

	is [TRUE]. Default [FALSE]
:FreeFormatOutput= [TRUE/FALSE]	When set to TRUE, causes output of the precompiler to be created in “free” source format.
:ImmediateCursor=[True/False]	(CitSQL for PostgreSQL Only) When set to True, causes as PREPARE EXEC to be executed before the OPEN when where a CURSOR is declared with OPEN and FETCH statements: SQL EXEC DECLARE xxx CURSOR … OPEN xxx FETCH .. INTO
:IncludeSearchPath=<Pathes>	Default [Current Working Directory] A comma, or semi-colon separated list of the directories in which CitSQL will look for Include files.
:LibName=<libname>	Defaults are: RCQMYSQL (MySQL), RCQPGSQL (PostgreSQL), RCQODBC (ODBC). The name of the RCQ runtime module. This option is needed only if you rename the RCQ runtime module. Default is: [FALSE]
:LogMode=[TRUE/FALSE]	When set to true the runtime component creates a log file called RCQDLL.log that traces all SQL operations.
:MaxMem=<number megabytes>	Default is: 100 Allocates <number megabytes> memory for the precompilation of very large source files.
:NoRecCode=<numeric>	Default is 1403 Allows mapping of value returned to indicate the end of a FETCH statement.
:Prefetch=<numeric>	Allows for the prefetch of <numeric> records in a network transaction.

:QuoteTranslation=<pattern>	Default is: QDB Allows mapping of single quotes, double quotes, and back quotes. By default, quotes are unchanged, which corresponds to a default value of QuoteTranslation=QDB.
:SelectPrepare=[True/False]	(CitSQL for PostgreSQL Only) Default is TRUE. Now, the preprocessor causes the PREPARE EXEC to be executed prior to the OPEN, and the results stored. When set to False, the former behavior is applied.
:StandardPrefix=<prefix>	Default is: [None] Characters prefixed to generated data items.
StepLimit=<numeric>	Default is: [None] Sets limit on the number of cases the CitSQL backtracking will consider. Required when you receive a Step Limit error.
StrictMode=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an SQL syntax in an EXEC statement.
:StrictPictureMode=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an PICTURE clause.
:TargetPattern=<pattern>	Default is: %d%r.cbp Describes a group of tokens that can be strung together components to describe the location and naming convention applied to the precompiled target file.

:TrimMode=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, alphanumeric (PIC X) strings that are passed to the database are first trimmed (right space removed) so that the actual data in the database does not have trailing spaces.
:TruncComments=[TRUE/FALSE]	Default is: [TRUE] When set to TRUE, Comments are truncated after column 72. When set to FALSE, comments are not truncated after column 72.
:UTFInput=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, Specifies that the source code contains literals encoded in UTF-8. When set to FALSE (the default), literals are treated as not being encoded in UTF-8.

Note- When setting Boolean command-line arguments, the words “True” and “False” can be written in upper or lower case, and their initials can be used as substitutes, as can equivalent integer codes. Thus, :DebugMode=TRUE, :DebugMode=True, :DebugMode=T, :DebugMode=1 are all equivalent.

Note- In Linux/Unix operating environments, the semi-colon should be avoided as a separator. White space should be avoided in directory names.

CloseOnCommit

:CloseOnCommit=[True/False]	The :CloseOnCommit option aligns CitSQL behavior with Oracle behavior in cases when a BEGIN … END construct contains an EXEC SQL SELECT statement followed by a COMMIT statement. When :CloseOnCommit=True, the preprocessor does not close the SQL cursor when the SELECT statement has executed. Instead, the cursor is closed after the COMMIT statement is

executed.

In a construct such as the BEGIN … END construct below, the default behavior is to open the SQL cursor before the execution of the SELECT statement, and to close the SQL cursor after the execution of the SQL statement. When :CloseOnCommit=True, the SQL cursor is closed after the COMMIT statement is executed.

```
BEGIN
EXEC SQL
SELECT aaa FROM X INTO :HostVar
END-EXEC
…
COMMIT
END
```

CursorSynteticName

:CursorSynteticName=[True/False]	Causes the cursor name to be generated at runtime by the RCQ runtime.

When "CursorSynteticName" is set to True, cursor names are generated at runtime by the RCQ runtime. This can be required if you use the same cursor name in several modules.

DBEncoding

:DBEncoding=<Valid codepage or UTF-8>	Specifies what encoding is used by the database storage.

IF a "DBEncoding" is defined :

- NATIONAL PIC N fields are converted from NATIONAL to "DBEncoding" encoding when writing to the database and from "DBEncoding" encoding to NATIONAL when reading from the database.

Valid codepages are those accepted by the COBOL intrinsic functions NATIONAL-OF and DISPLAY-OF. For a complete list of the valid codepages recognized by the COBOL-IT compiler, run the command:

```
> cobc –list-codepage
```

DeallocateCloseCursor

:DeallocateCloseCursor=[True/False]	Causes the CLOSE cursor to also deallocate the DECLARED cursor.

Normally when a program executes an OPEN Cursor, the RCQ runtime makes a DECLARE Cursor the first time the OPEN is executed. Then, all subsequent executions of the same OPEN will reuse

the previously declared cursor. In situations where several COBOL programs use the same CURSOR name or when using CANCEL, this can result in the return of the POSTGres error: "Prepared name xxx already exists" even if the cursor xxx is closed.

To fix this, use this flag so that the DECLAREd cursor is deallocated when the CLOSE cursor statement is executed.

DefaultCCSID

:DefaultCCSID=<Valid codepage or UTF-8>	Specifies the default CCSID for string fields that have no explicit CCSID declaration.

Valid codepages are those accepted by the COBOL intrinsic functions NATIONAL-OF and DISPLAY-OF. For a complete list of the valid codepages recognized by the COBOL-IT compiler, run the command:

> cobc –list-codepage

Different CCSIDs can be declared for different fields in the source code. The CCSID of individual fields may be explicitly declared using the IBM syntax:

Example:
EXEC SQL DECLARE :<HOSTVARIABLE> VARIABLE CCSID <Valid codepage or UTF-8> END-EXEC.

Example (multiple Host variables):
EXEC SQL DECLARE :<HostVar1>, :<HostVar2> VARIABLE CCSID <Valid codepage or UTF-8> END-EXEC.

If a CCSID is defined (by default or explicitly for the field)

- STRING fields (PIC X, non-numeric USAGE DISPLAY fields and PIC N fields) are converted from "CCSID" encoding To "DBEncoding" encoding when writing to the database and from "DBEncoding" encoding to "CCSID" encoding when reading from the database.

- Fields described with the PIC N clause are considered to be declared as USAGE NATIONAL. Note- For fields described as PIC N VARCHAR, the length of the -LEN fields is given in number of characters (not in number of bytes). For clarification, in this example, the length of « 12345 » in FIELDA-ARR is 5, while the number of bytes in storage is 10:

  01 FIELDA PIC N(100) VARCHAR.

  MOVE « 12345 » TO FIELDA-ARR
  MOVE 5 TO FIELDA-LEN

- For fields described with the PIC N clause, CCSID values 1200, 1201 and UTF-16BE are the only accepted values. Other values provoke a warning and are ignored.

DebugMode

:DebugMode= [TRUE/FALSE]

Default [FALSE]
When set to TRUE, causes log file created when LogMode=TRUE to contain more detail in some situations.

When set to TRUE, and when LogMode set to TRUE, the runtime logfile RCQDLL.log will contain more detailed information about host variables. Set DebugMode to True to maximize the information returned from CALLs to the runtime component.

ExpandIncludes

:ExpandIncludes= [TRUE/FALSE]

Default [FALSE]
When set to TRUE, causes all COPY files in the program to be expanded into the source file prior to pre-compiling the source.

When set to TRUE, COPY files are expanded in the source file prior to the pre-compiling of the source. Note- If any SQL statements reference a field in a COPY book, the the command-line option :ExpandIncludes= TRUE is required.

ForceStringMode

ForceStringMode=[TRUE/FALSE]

Default is [FALSE] unless :DBEncoding is set to UTF-8 or utf8. Then, the default is True.

When set to True, CitSQL non-compound PIC X data fields are sent to the database as a C-String (where the String is terminated by the character X'00'). When set to FALSE, CitSQL sends PIC X data to the database as a byte-array.

The :ForceStringMode option affects whether a PIC X data field is sent to the database as a byte-array, or as a C-String.

When set to False, CitSQL sends PIC X data to the database as a byte-array. This is done to respect the COBOL semantic that does not consider the character X'00' as the end of a string. This can be a problem if the user passes a host variable describe with PIC X to a field declared as a DATE, for example. DATEs are stored as 4-byte INTs by Postgres, so Postgres expects the DATE to have a length of 4-bytes.

When set to True, CitSQL sends PIC X data to the database as a C-String (where the String is

terminated by the character X'00'). In this case, Postgres can convert the C-String into a DATE. This is not activated by default, as it is counter to the pure COBOL semantic.
:ForceStringMode=TRUE only applies to elementary data items. Group items containing sub-items are not affected by this setting.

Rules:

- When :DBEncoding=UTF-8 or utf8 :ForceStringMode is set to True as UTF-8 does not allow the X'00' character.
- :ForceStringMode=True only affect the behavior of non-compound fields. That is,
  In this case:
  01 COL1 PIC X(10).

  COL1 will be send a C-String

  However, in this case:
  01 GRP1.
  02 COL11 PIC X(10)
  …
  GRP1 will always be sent as a byte-array whatever the :ForceStringMode parameter is.

FreeFormatOutput

:FreeFormatOutput= [TRUE/FALSE]	Default [FALSE] When set to TRUE, causes output of the precompiler to be created in "free" source format.

The FreeFormatOutput option, when set to TRUE, causes the ouput of the precompiler to be created in "free" source format. Free source format, or "terminal format" applies different rules for establishing the location of the Area A, the Indicator Area, Area B, and the Identification Area. COBOL-IT source programs that are written in Terminal Format must be compiled with the –free compiler flag. For more details on the free source format, see Source Code Management Topics in the Reference Manual.

ImmediateCursor

:ImmediateCursor=[True/False]	(CitSQL for PostgreSQL Only) When set to True, causes as PREPARE EXEC to be executed before the OPEN when where a CURSOR is declared with OPEN and FETCH statements: SQL EXEC DECLARE xxx CURSOR … OPEN xxx FETCH .. INTO

The results of the PREPARE EXEC are stored, and returned by the FETCH.

Attention should be taken when applying the ImmediateCursor preprocessor parameter. Since the full results of the cursor are returned before the OPEN, this parameter should only be applied for cursors returning small numbers of lines.

When not using the ImmediateCursor precompiler option, you can cause the PREPARE EXEC to be executed prior to the OPEN for cursors using the statement :

SQL EXEC DECLARE xxx STATIC CURSOR …

IncludeSearchPath

:IncludeSearchPath=<Pathes>	Default [Current Working Directory] A comma, or semi-colon separated list of the directories in which CitSQL will look for Include files.

The IncludeSearchPath option is expressed as a list of directories, separated by commas, or semi-colons, in which CitSQL searches for Include files.

Note- The CitSQL precompiler must locate the file SQLCA.cpy fle, which, after installation, is installed by default in the PreprocessorDistrib/include directory.

As a result, the basic precompiler command line is:

citsql :IncludeSearchPath=[path to SQLCA] [source file]

In an example, where the SQLCA.cpy file is located in a subdirectory called ./include, and the source file is called prog1.cob, the command line would be:

citsql :IncludeSearchPath=./include prog1.cob

LibName

:LibName=<libname>	Defaults are: RCQMYSQL (MySQL), RCQPGSQL (PostgreSQL), RCQODBC (ODBC). The name of the RCQ runtime module. This option is needed only if you rename the RCQ runtime module.

As delivered, the runtime component name corresponds to the default settings for LibName. If you wish to rename these libraries, you must use LibName to prompt the precompiler to change the target of the CALL statement in the generated code.
Using the sample program provided with the distribution as an example, note how the EXEC SQL CONNECT statement is translated by the CitSQL for PostgreSQL precompiler. In the example below, the procompiler is generating code using the default LibName setting of RCQPGSQL:

```
EXEC SQL
```

```
    CONNECT TO         :WRK-DB
                USER   :WRK-UNAME
                USING  :WRK-HOST
          IDENTIFIED BY:  WRK-PASSWORD
```

Is translated to:

```
*EXEC SQL
*    CONNECT TO         :WRK-DB
*                USER   :WRK-UNAME
*                USING  :WRK-HOST
*          IDENTIFIED BY:  WRK-PASSWORD

IF 1=1 THEN
               CALL "RCQPGSQL" USING BY VALUE RCQ_C_CONN
                                             BY REFERENCE RCQ-CFG
                                             BY REFERENCE SQLCA
                 BY REFERENCE WRK-UNAME OF WRK-VARS RCQ-DESC-25
                 BY REFERENCE WRK-PASSWD OF WRK-VARS RCQ-DESC-25
                 BY REFERENCE WRK-HOST OF WRK-VARS RCQ-DESC-25
                 BY REFERENCE WRK-DB OF WRK-VARS RCQ-DESC-25
                 BY REFERENCE RCQ-NULL-VAR RCQ-NULL-VAR-DESC
 END-IF
```

LogMode

:LogMode=[TRUE/FALSE]

Default is: [FALSE]
When set to TRUE the runtime component creates a log file called RCQDLL.log that traces all SQL operations.

When set to TRUE, causes a log file called RCQDLL.log to be created, which contains a trace of all CALLs to the Runtime Component.
Observe below an excerpt of the output of cit_test when run with LOGMODE=True:

```
[Wed 2013-04-24 17:03:39]
[Wed 2013-04-24 17:03:39][EXECUTE QUERY]
[Wed 2013-04-24 17:03:39][statement: DROP TABLE TAB_1]
[Wed 2013-04-24 17:03:39][nbre of param : 0]
[Wed 2013-04-24 17:03:39]
[Wed 2013-04-24 17:03:39][MYSQL_STMT_QUERY] OK
[Wed 2013-04-24 17:03:39]
[Wed 2013-04-24 17:03:39][EXECUTE PREPARE]
[Wed 2013-04-24 17:03:39][statement: CREATE TABLE TAB_1(COL_1
CHAR(20),COL_2 FLOAT,COL_3 INT)]
[Wed 2013-04-24 17:03:39][nbre of param : 0]
[Wed 2013-04-24 17:03:39]
[Wed 2013-04-24 17:03:39][MYSQL_STMT_QUERY] OK
```

```
[Wed 2013-04-24 17:03:39]
[Wed 2013-04-24 17:03:39][EXECUTE PREPARE]
[Wed 2013-04-24 17:03:39][statement: INSERT into
TAB_1(COL_1,COL_2,COL_3)VALUES(?,?,?)]
[Wed 2013-04-24 17:03:39][nbre of param : 3]
[Wed 2013-04-24 17:03:39][MYSQL_STMT_INIT] OK
```

MaxMem

:MaxMem=[Number]

Default is: 100
Allocates <number megabytes> memory for the precompilation of very large source files.

Allocates <number megabytes> memory for the precompilation of very large source files.
To allocate 200 MB for the precompilation of a file, add the line

:MaxMem=200

To your stdoptions.res file.

By default, CitSQL will allocate 100MB of memory for the precompilation of a source file. In most cases, this will be more than adequate. If this default memory allocation is too small, CitSQL will fail, and abort, and you should increase the setting of MaxMem. If you provide a setting that is in excess of what the system can allocate, this will also cause CitSQL to fail and abort.

NoRecCode

:NoRecCode=<numeric>

Default is 1403
Allows mapping of value returned to indicate the end of a FETCH statement.

The :NoRecCode option sets the value returned to indicate the end of a FETCH statement, when no more records are found. The default value is 1403, which corresponds to the Oracle SQLCODE for a NOT FOUND condition.

Prefetch

Allows for the prefetch of <numeric> records in a network transaction, where <numeric> is a whole number that represents the number of records to read in a networked transaction..

Prefetch can improve performance where networked transactions are concerned.
The Prefetch=<numeric> option can either be implemented on the PGSQL command line, or dynamically by populating the RCQ-CFG-PREFETCH variable before performing an EXEC-SQL FETCH statement. As an example of a dynamic implementation of the prefetch functionality:

MOVE <numeric> TO RCQ-CFG-PREFETCH

…
EXEC-SQL FETCH …

Where <numeric> is a whole number that represents the number of records to read in a networked transaction.

The Prefetch option is available only with PGSQL.

Quote Translation

:QuoteTranslation=<pattern>	Default is: QDB Allows mapping of single quotes, double quotes, and back quotes. By default, quotes are unchanged, which corresponds to a default value of QuoteTranslation=QDB.

Quoting is an area where the various database engines can have fairly different behaviours. Quotes (of some form) are used to specify string literals and to refer to SQL elements (tables, columns) which name clash with a reserved words – when a table is named "SELECT", for instance. Depending on the database engine at hand, different quotes are used for these different purposes. When writing new applications, it is of course always possible to use the quotes according to the target database's rules at hand. However, things are trickier when dealing with a legacy applications, where a given quoting convention is used and trying to use this application on another database which then requires a different quoting convention.
Of course, one can always modify the programs manually to comply with this different quoting convention, but CitSQL provides a more comfortable mechanism.
One can have the quotes remapped at compile time – hence, without any negative impact on runtime performance – so that the actual quotes in the program's SQL statements are translated before being sent to the database engine.
This quoting replacement process is controlled by the `:QuoteTranslation` command-line option. It takes a three character string argument, made of "Q", "S", "D", and "B".
The first character of this argument denoted what simple quotes must be mapped to. The second denotes what denotes what double quotes must be mapped to, while the third denotes what the backquote must be mapped to.
When defining these mapping, the following codes are used:

- 'S' or 'Q' for single quotes
- 'D' for double quotes
- 'B' for backquotes.

For instance, to have all quotes mapped to double quotes, one can use a command-line option such as `:QuoteTranslation=DDD`. Similarly, to swap single and double quotes, one can use an option such as `:QuoteTranslation=DQB`.
Since CitSQL's default behaviour is to leave the quotes as they are expressed explicitly in the source code, the default value for this command-line option is "QDB".

SelectPrepare

:SelectPrepare=[True/False] (CitSQL for PostgreSQL Only)
Default is TRUE. Now, the preprocessor causes the PREPARE EXEC to be executed prior to the OPEN, and the results stored. When set to False, the former behavior is applied.

Affects the code generated by the EXEC SQL SELECT …. INTO …

Previously, the preprocessor generated a CURSOR > OPEN > 1 FETCH > CLOSE sequence. Now, the preprocessor causes the PREPARE EXEC to be executed prior to the OPEN, and the results are stored. The result is a significant improvement in performance.

Setting :SelectPrepare=False causes the preprocessor to apply the former behavior.

StandardPrefix

:StandardPrefix=<prefix> Default is: [None]
Characters prefixed to generated data items.

A number of data items are automatically generated by the precompiler.
As an example – SQL-VARS is a group item that is automatically generated. When StandardPrefix=x, the SQL-VARS group item is generated as x-SQL-VARS. This device can be used to avoid collisions between variables automatically generated by the precompiler and variables that exist in the source code.

StepLimit

:StepLimit=<numeric> Default is: [None]
Sets limit on the number of cases the CitSQL backtracking will consider. Required when you receive a Step Limit error.

The CitSQL parser is based on a Backtracking technology. In order to do this, it must set a limit on the number of cases it must be able to consider. You can control this limit with the :StepLimit option. Normally you will not need to use the :StepLimit option. However if, when pre-compiling your source code, you receive a Step limit parse error, as in the example below, then you should include the :StepLimit option in your pre-compilation command, and increase the value above the default.

As an example:
citpgsql :IncludeSearchPath=/opt/distribPGSQL-64/include :StepLimit=2000000 ./src /testit.cbl

Example of Step limit parse error:
1:./src /testit.cbl: Step limit parse error: ./src/testit.cbl

StrictMode

:StrictMode=[TRUE/FALSE]

Default is: [FALSE]
When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an SQL syntax in an EXEC statement.

When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an SQL syntax in an EXEC statement.

StrictPictureMode

:StrictPictureMode=[TRUE/FALSE]

Default is: [FALSE]
When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an PICTURE clause.

When set to TRUE, the CitSQL precompiler aborts in cases where it does not recognize an PICTURE clause.

TargetPattern

:TargetPattern=<pattern>

Default is: %d%r.cbp

Describes a group of tokens that can be strung together components to describe the location and naming convention applied to the precompiled target file.

The TargetPattern option involves stringing together a number of components to establish the location, and naming convention to be applied to target (precompiled) files. It is recommended that when using the TargetPattern option, a resource file be used.
TargetPattern takes as an argument a string of components, which represent various aspects of the the naming and location of the target file.

The codes recognized by the TargetPattern option are:

Code	Description
%p	Pathname of target file derived from source file
%f	Filename of target file derived from source file, without explicit directory name
%r	Default is: %r.cbp Radical, file name of target file without extension derived from source file. The usage

	of the %r pattern is typically followed by the desired extension. Example: `:TargetPattern=%r.cbl`
%e	Extension for the target file to be added to the end of the source file
%d	Default is: Directory of Source file. Directory of target file same as directory of source file.

Note that the default TargetPattern applied by CitSQL is %d%r.cbp

The result is that the simple command line:

>citsql test.cob produces an output file called test.cbp in the same directory.

TrimMode

:TrimMode=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, alphanumeric (PIC X) strings that are passed to the database are first trimmed (right space removed) so that the actual data in the database does not have trailing spaces.

When set to TRUE, alphanumeric (PIC X) strings that are passed to the database are first trimmed (right space removed) so that the actual data in the database does store character strings with trailing spaces.

TruncComments

:TruncComments=[TRUE/FALSE]	Default is: [TRUE] When set to TRUE, Comments are truncated after column 72. When set to FALSE, comments are not truncated after column 72. When set to TRUE, Comments are truncated after column 72.

UTFInput

:UTFInput=[TRUE/FALSE]	Default is: [FALSE] When set to TRUE, Specifies that the source code contains literals encoded in UTF-8. When set to FALSE (the default), literals are treated as not being encoded in UTF-8.

Usage with a resource file

Storing multiple options in a resource file

When using multiple options, it is best to store the options in a resource file. In a resource file, options are stored on separate lines. Empty lines, and lines beginning with a "#" or ";" character are ignored.

When invoking a resource file from the command line, the resource file is prefixed with an "@" character.

Enter the options on separate lines, as follows:

```
:IncludeSearchPath=../../include
:TargetPattern=%d%r.cbl
```

Save the file as stdoptions.res.

Then, reference the resource on the command line as follows:

```
>citsql @stdoptions.res prog1.cbl prog2.cbl
```

Storing multiple programs in a resource file

Multiple programs may also be stored in a resource file, and invoked from the command line by prefixing the resource file with an "@" character.

Enter the programs on separate lines, as follows:

```
prog1.cbl
prog2.cbl
```

Save the file as stdprogs.res.

Then, reference the resource on the command line as follows:

>citsql @stdoptions.res @stdprogs.res

Using Wildcards

Wildcards are not expanded by the precompiler. However, in Linux/Unix environments, wildcard expansions are performed by the shell.

Thus, the command:

>citsql @stdoptions.res *.cob

Would have the effect of causing each of the files with the .cob extension located in the current directory to be passed through the precompiler.

Runtime Environment Variables

RCQLOG=<logfile>

The RCQLOG runtime environment variable allows a filename to be associated with the logfile produced by CitSQL when precompilig with :LogMode=True

<logfile> can include a complete path name followed by a file name. In the absence of a complete path name, the file name will be applied, and the file will be generated in the current working directory.

When the RCQLOG runtime environment variable is not set, the logfile is named RCQDDL.LOG, and created in the current working directory. The RCQLOG runtime environment variable must be set prior to the use of CitSQL.

Note that when using :LogMode=True, the runtime logfile will contain more detailed information about host variables when also using :DebugMode=True.

Running the sample test program

The `samples\cobol` directory of the distribution contains the sample programs dbconnect.cob and cit_test.cob. dbconnect.cob is the simpler of the two programs, and just demonstrates that you have established a connection to the database.

cit_test.cob requires a username, password, database name and host name. Open the source file in a text editor, and provide a valid username, password, and database name for your database.

For guidelines on creating a username, password, database, and host name in your database, see Appendix 1- Addressing the various databases.

Provided with these, cit_test.cob checks for a table TAB_1. If the table exists, it drops the table. It then creates a table TAB_1, writes to it, reads the data back, disconnects, and does a final read to

test an error condition.

Precompile the Sample Test Programs

Commands for precompiling the sample programs are contained in the batch file compil.bat:

For MySQL :

SET PATH=..\..\bin;%PATH%

citmysql :IncludeSearchPath=../../include cit_test.cob

citmysql :IncludeSearchPath=../../include dbconnect.cob

For PostgreSQL :

SET PATH=..\..\bin;%PATH%

citpgsql :IncludeSearchPath=../../include cit_test.cob

citpgsql :IncludeSearchPath=../../include dbconnect.cob

For ODBC:

SET PATH=..\..\bin;%PATH%

citodbc :IncludeSearchPath=../../include cit_test.cob

citodbc :IncludeSearchPath=../../include dbconnect.cob

Compile the Sample Test Programs

Commands for compiling the sample programs are contained in the batch file compil.bat:

Dbconnect.cbp is compiled with no compiler flags, creating a shared object, or DLL in Windows. Cit_test.cbp is compiled with the –x compiler flag, demonstrating how to create an executable object. The –x compiler flag is not required.

cobc dbconnect.cbp

cobc -x cit_test.cbp

Run the Sample Test Programs

Commands for running the sample programs are contained in the batch file run.bat:

Since cit_test has been compiled with the –x compiler flag, it is a native executable.

Running dbconnect requires the COBOL-IT runtime, cobcrun, as it has been created as a shared library/DLL,

SET COB_LIBRARY_PATH=.;..\..\bin
cit_test

or

SET COB_LIBRARY_PATH=.;..\..\bin

cobcrun dbconnect

The SQL Communications Area (SQLCA)

The SQLCA is the data structure through which Warnings and Errors are returned from the database to the running application.

Useful Guidelines are:

SQLCODE contains the status-code returned by the most recently executed SQL statement.

SQLERRM contains the length of the error message returned by the most recently executed SQL statement in SQLERRML, and the text of the error message returned by the most recently executed SQL statement in SQLERRMC. Error messages cannot greater than 70 bytes in length are truncated.

SQLERRD contains status codes returned by the database.

SQLWARN contains warning flags.

SQLSTATE contains a return code that can be referenced to determine whether an SQL statement executed successfully, and if not, the type of error that has occurred.

For details, refer to your database documentation.

The SQLCA Data Structure

```
01 SQLCA.
    05  SQLCAID          PIC X(8)          VALUE "SQLCA   ".
    05  SQLCABC          PIC S9(8) COMP-5 VALUE 136.
    05  SQLCODE          PIC S9(8) COMP-5 VALUE 0.
    05  SQLERRM.
        10  SQLERRML     PIC S9(4) COMP-5.
        10  SQLERRMC     PIC X(70).
    05  SQLERRP          PIC X(8).
    05  SQLERRD          PIC S9(8) COMP-5 OCCURS 6.
    05  SQLWARN.
        10  SQLWARN0     PIC X.
        10  SQLWARN1     PIC X.
        10  SQLWARN2     PIC X.
        10  SQLWARN3     PIC X.
        10  SQLWARN4     PIC X.
        10  SQLWARN5     PIC X.
        10  SQLWARN6     PIC X.
        10  SQLWARN7     PIC X.
        10  SQLWARN8     PIC X.
        10  SQLWARN9     PIC X.
        10  SQLWARN10    PIC X.
        10  SQLWARNA     PIC X REDEFINES SQLWARN10.
    05  SQLSTATE         PIC X(5).
```

```
05  FILLER          PIC X(21).
```

Support for USAGE Varchar, Varraw, Varying

CitSQL supports the non-COBOL USAGE Clauses:

USAGE VARCHAR
USAGE LONG VARCHAR
USAGE VARRAW
USAGE LONG VARRAW
USAGE VARYING.

When the CitSQL Precompiler encounters one of these clauses, the USAGE clause is commented out and replaced with a group item of the same name, that consists of two elementary items, one that contains the "-LEN" suffix, and one that contains the "-ARR" suffix.

As an example :

05 COL-5 PIC X(50) USAGE VARCHAR.

Is converted to:

```
*****RC*SQL**************
* 05 COL-5 PIC X(50) USAGE VARCHAR. *****RC*SQL**************
05 COL-5.
  06 COL-5-LEN PIC S9(4) COMP-5 VALUE 0.
  06 COL-5-ARR PIC X(50).
*****RC*SQL**************
```

Conversion details of these non-COBOL data types are expanded on below:

Usage Clause	Used for	"-LEN" field
[LONG] VARCHAR	For text, or binary fields, possible storage conversion, and trim	PIC S9(4) COMP-5.
VARYING	For text, or binary fields, possible storage conversion, and trim	PIC S9(4) COMP-5.
[LONG] VARRAW	For binary fields, no conversion	PIC S9(9) COMP-5.

The best matching database declaration for these data types depends on the database being used. For guidelines see the table below:

USAGE Data Type	MySQL Database Declaration	PostGreSQL Database Declaration	SQL Server Database Declaration
VarChar/Varying	VARCHAR	VARCHAR	VARCHAR

Long VarChar	TEXT	VARCHAR	VARCHAR
VarRaw	VARBINARY	BYTEA	VARBINARY
Long VarRaw	BLOB	BYTEA	VARBINARY

Addressing the various databases

Connection

Every database has its own syntax for connecting to an instance. CitSQL recognizes four components:

- The user name
- The password
- The database instance
- The hostname

The sample below show various syntactical forms that can be used to combine these elements. Please note that these elements can be present as references to variables or as string literals.

```
EXEC SQL
  CONNECT                :WRK-UNAME
          IDENTIFIED BY  :WRK-PASSWD
END-EXEC
```

```
EXEC SQL
  CONNECT TO             :WRK-DB
          USER           :WRK-UNAME
          IDENTIFIED BY  :WRK-PASSWD
END-EXEC
```

```
EXEC SQL
  CONNECT TO             :WRK-DB
          USER           :WRK-UNAME
          USING          :WRK-HOST
          IDENTIFIED BY  :WRK-PASSWD
END-EXEC
```

```
EXEC SQL
  CONNECT                :WRK-UNAME
          IDENTIFIED BY  :WRK-PASSWD
          AT             :WRK-DB
END-EXEC
```

Addressing MySQL

Preparing the sample program

The following COBOL connection can be used

```
EXEC SQL
  CONNECT TO              :WRK-DB
          USER            :WRK-UNAME
          USING           :WRK-HOST
          IDENTIFIED BY :WRK-PASSWD
END-EXEC
```

Where

- :WRK-DB is DBName
- :WRK-UNAME is DbUserName
- :WRK-HOST is “DbServer:PortNr”
- :WRK-PASSWD is DbPassword

The PortNr options is optional. By default, it use the default port (3306 for MySQL)

As an example:

In the CitSQL distribution… ~/samples/cobol/dbconnect.cob

```
01  WRK-VARS.
   ******** INFORMATION TO CONNECT TO THE DB*******************
            05 WRK-UNAME  PIC X(80) VALUE "testuser".
            05 WRK-PASSWD PIC X(80) VALUE "test".
            05 WRK-DB     PIC X(80) VALUE "dbu".
            05 WRK-HOST   PIC X(80) VALUE "localhost".
   ************************************************************
   . . .
    PROCEDURE DIVISION.
   ***** THE COMPLETE CONNECTION SEE THE MANUAL FOR OTHER
        EXEC SQL
          CONNECT TO            :WRK-DB
                  USER          :WRK-UNAME
                  USING         :WRK-HOST
                  IDENTIFIED BY :WRK-PASSWD
       END-EXEC
```

Preparing the conection to the database

From the Command Line, MySQL recognizes the following commands:

mysql --user=DbUserName --password=DbPassword –host=DbServer
--port=portnr DBName

SQL Statement	Referencing variable	Set up
CONNECT TO :WRK-DB	05 WRK-DB PIC X(80) VALUE "dbu".	Create database "dbu" on localhost
USER :WRK-UNAME	05 WRK-UNAME PIC X(80) VALUE "testuser".	Create user "testuser" with password "test", and privileges to log on to "dbu", create tables, read and write data
USING :WRK-HOST	05 WRK-HOST PIC X(80) VALUE "localhost".	See above. Our server is set up as "localhost"
IDENTIFIED BY :WRK-PASSWD	05 WRK-PASSWD PIC X(80) VALUE "test".	Create password "test" for user "testuser"

Addressing PostgreSQL

Preparing the sample program

The following COBOL connection can be used

```
EXEC SQL
  CONNECT TO              :WRK-DB
          USER            :WRK-UNAME
          USING           :WRK-HOST
          IDENTIFIED BY :WRK-PASSWD
END-EXEC
```

Where

- :WRK-DB is "DBName"
- :WRK-UNAME is "DbUserName"
- :WRK-HOST is "DbServer:PortNr"
- :WRK-PASSWD is "DbPassword"

The PortNr options is optional. By default, it use the default port (5432 for PostgreSQL)

As an example:

In the CitSQL distribution… ~/samples/cobol/dbconnect.cob

```
01  WRK-VARS.
    ******** INFORMATION TO CONNECT TO THE DB*******************
             05 WRK-UNAME  PIC X(80) VALUE "testuser".
             05 WRK-PASSWD PIC X(80) VALUE "test".
             05 WRK-DB     PIC X(80) VALUE "dbu".
             05 WRK-HOST   PIC X(80) VALUE "localhost".
    ****************************************************************
    . . .
     PROCEDURE DIVISION.
    ***** THE COMPLETE CONNECTION SEE THE MANUAL FOR OTHER
         EXEC SQL
           CONNECT TO            :WRK-DB
                   USER          :WRK-UNAME
                   USING         :WRK-HOST
                   IDENTIFIED BY :WRK-PASSWD
        END-EXEC
```

Preparing the conection to the database

From the Command Line, PostgreSQL recognizes the following commands:

Psql –d DBName –U DbUserName –h DbServer –w

PostgreSQL also recognizes the following string:

Server=DbServer;Port=PortNr;Database=DBName;User Id=DbUserName; Password=DbPassword;

SQL Statement	Referencing variable	Set up
CONNECT TO :WRK-DB	05 WRK-DB PIC X(80) VALUE "dbu".	Create database "dbu" on localhost
USER :WRK-UNAME	05 WRK-UNAME PIC X(80) VALUE "testuser".	Create user "testuser" with password "test", and privileges to log on to "dbu", create tables, read and write data
USING :WRK-HOST	05 WRK-HOST PIC X(80) VALUE "localhost".	See above. Our server is set up as "localhost"
IDENTIFIED BY :WRK-PASSWD	05 WRK-PASSWD PIC X(80) VALUE "test".	Create password "test" for user "testuser"

Addressing Microsoft SQL Server (ODBC)

Preparing the Sample Program

Databases supporting ODBC use a string to establish a connection. The following example

"Driver={SQL Server Native Client 10.0};
Server=MyServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;"
shows how to connect to the Microsoft SQL Server ODBC driver.

As an example:

```
       PROCEDURE DIVISION.
      ***** THE COMPLETE CONNECTION SEE THE MANUAL FOR OTHER
          STRING "Driver={SQL Server};Server=DJS-VAIO\SQLEXPRESS;"
                 "Database=dbu;Uid=testuser;Pwd=test"
                DELIMITED BY SIZE
                INTO WRK-ODBC
```

```
.....
    EXEC SQL
        CONNECT TO      :WRK-ODBC
    END-EXEC
```

For more connection string samples, please visit the following web sites:
http://www.connectionstrings.com/sql-server-2005
http://www.connectionstrings.com/sql-server-2008

The following COBOL connection can be used.

```
EXEC SQL
  CONNECT TO              :ODBC-DB
END-EXEC
```

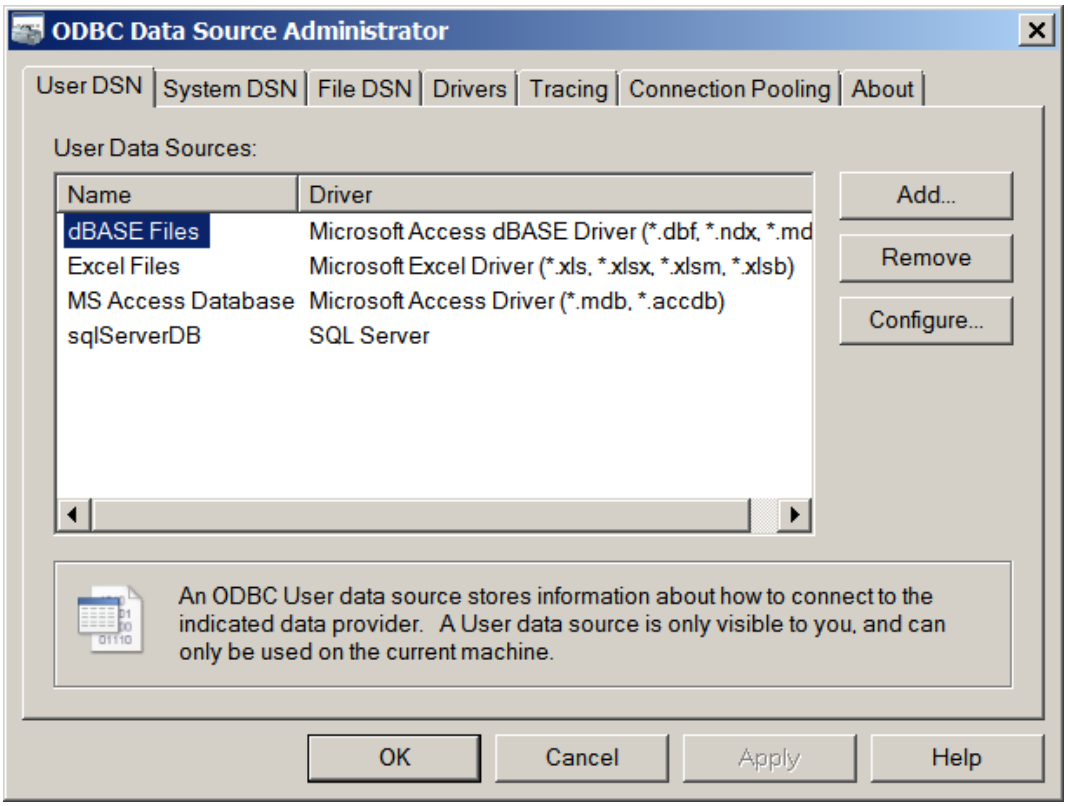
Where

- :ODBC-DB is the ODBC string

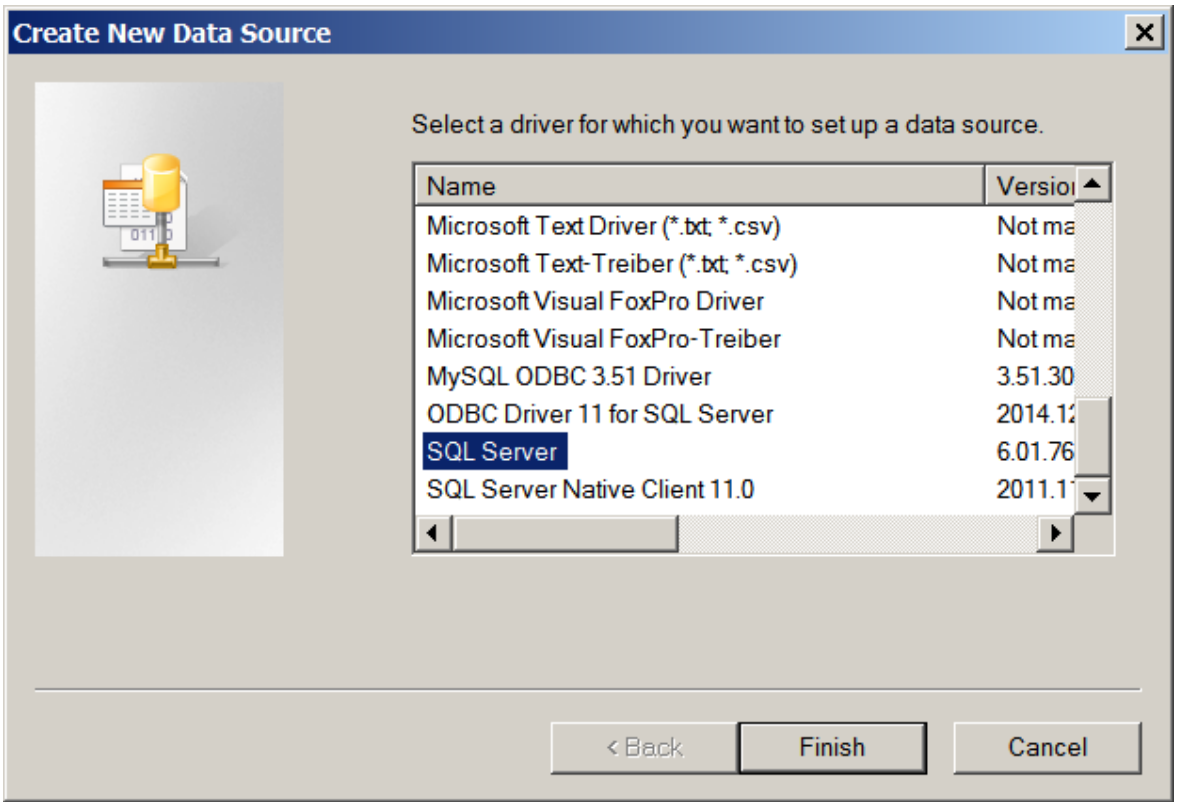
Limitations
The limitations are those of the ODBC drivers.

Preparing the connection to the database

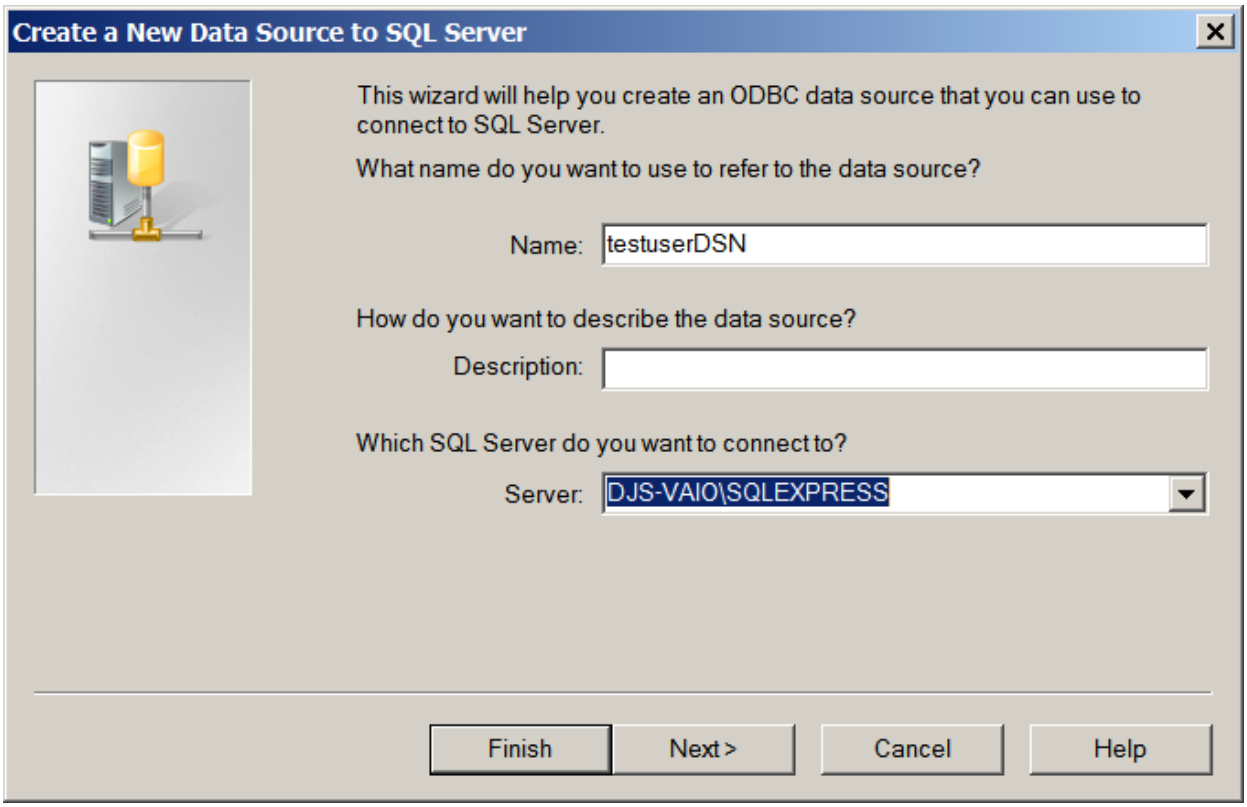
The ODBC Data Source Administrator



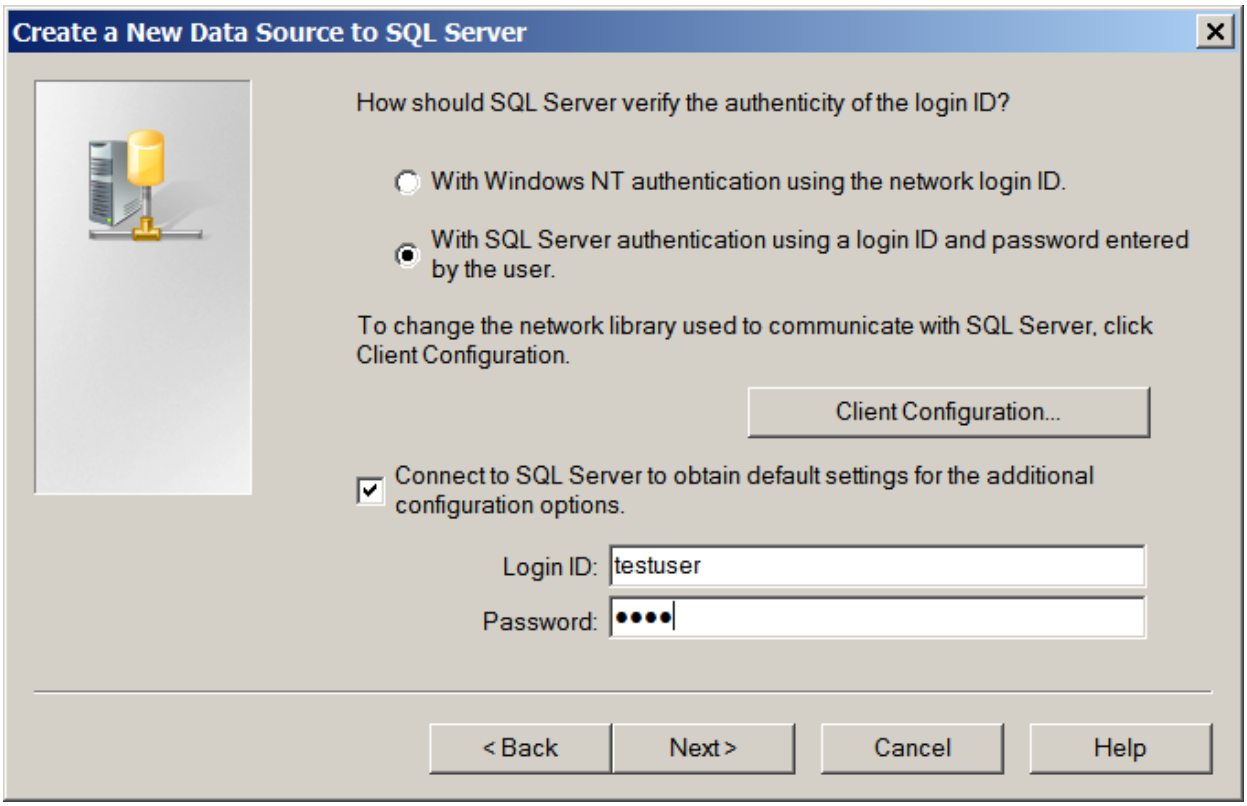
Create a New Data Source



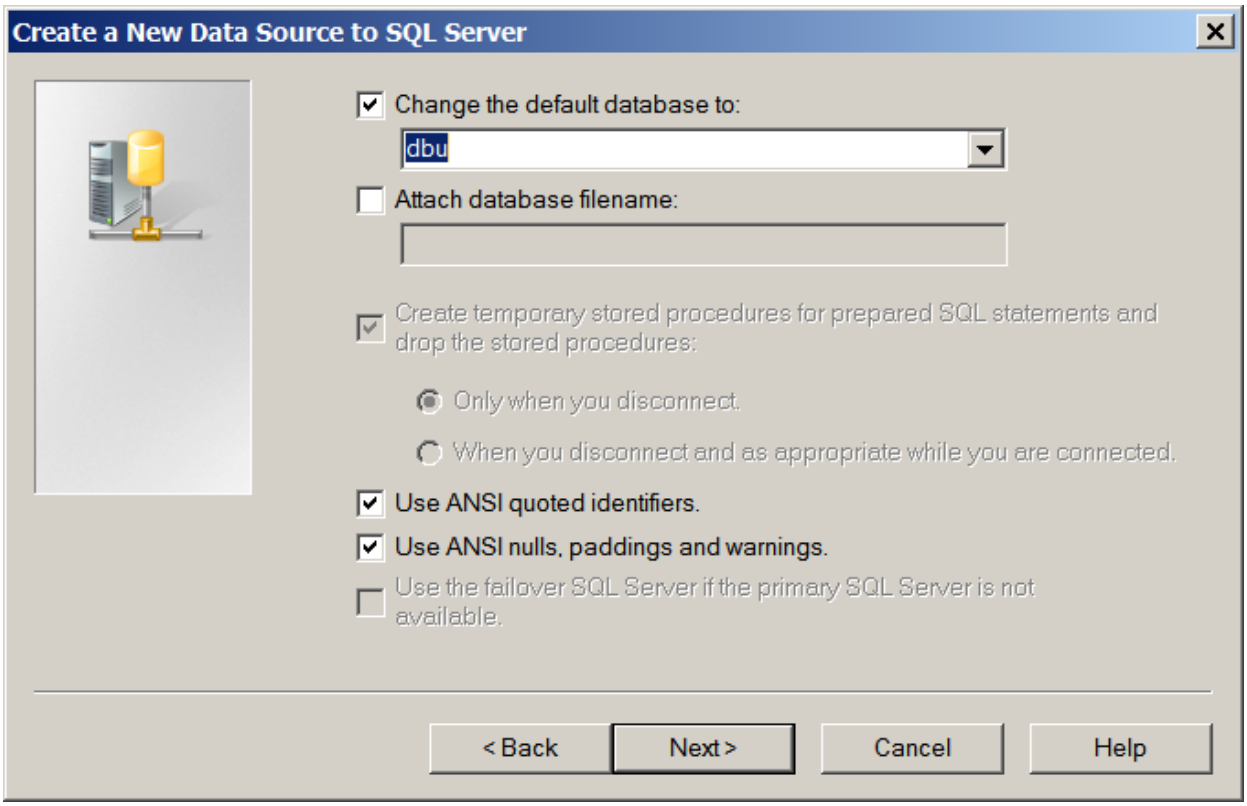
Create a new Data Source to SQL Server



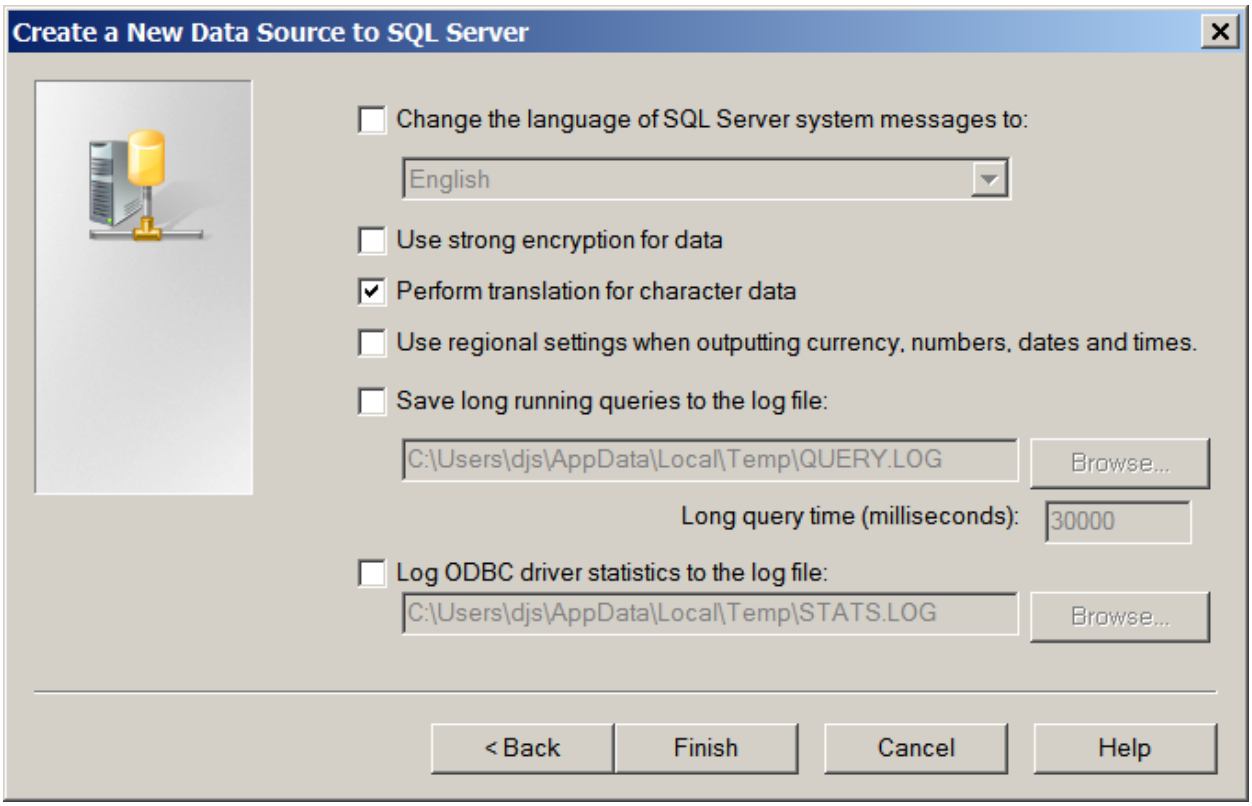
Create a new Data Source to SQL Server (continued)



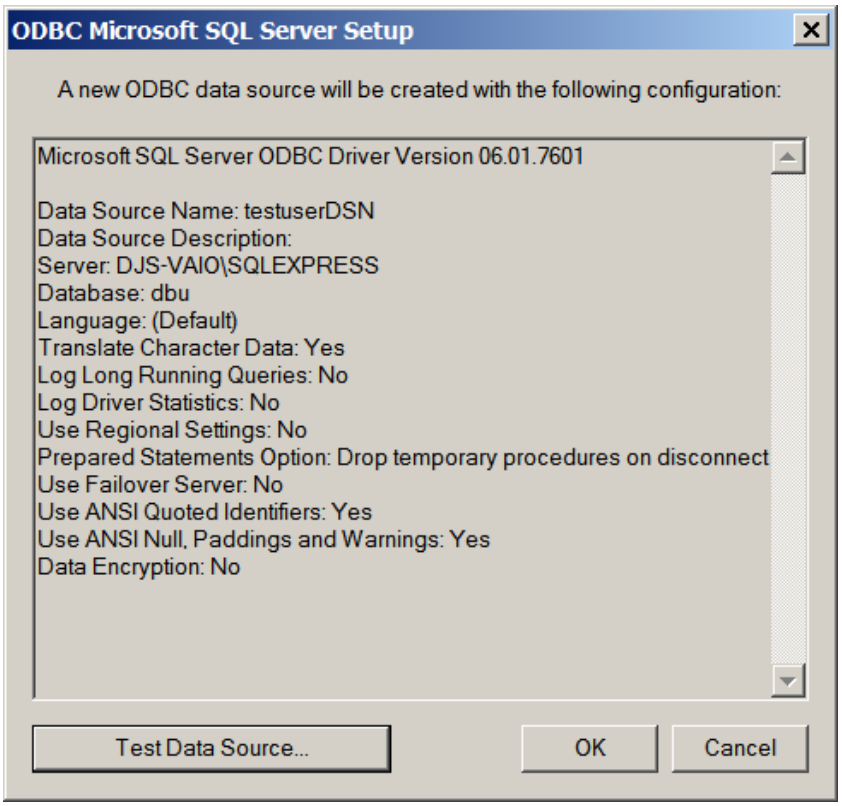
New Data Source to SQL Server



New Data Source to SQL Server (continued)



ODBC SQL Server Setup



SQL Server ODBC Data Source Test

Dealing with unrecognized SQL

When CitSQL encounters an SQL statement if cannot parse, two actions can be taken:

- If strict mode is enabled, an error message is produced and precompilation is aborted
- If strict mode is not enabled, CitSQL will see whether the SQL statement refers to any host variable. If it does not, it will send the SQL statement as is, and it will then be up to the database at hand to report errors if any.

This facility could not be extended to cases where host variables are being used because the treatment of these variables depends on whether they are used in read or write mode, and there is no way one can discriminate between these two when dealing with an unrecognized SQL statement.

This tolerance for unrecognized SQL statements can prove especially useful when dealing with DDL statements, where each database supports its own set of extensions to deal with table spaces, indexing methods, special column types, etc.

Using CitSQL in the Developer Studio

Using CitSQL for MySQL in the Developer Studio

Verify the setup of the database

>mysql --user=testuser --host=localhost --port=3306 --execute="showdatabases;" --password

After entering the password, you should see that dbu is listed as a database.

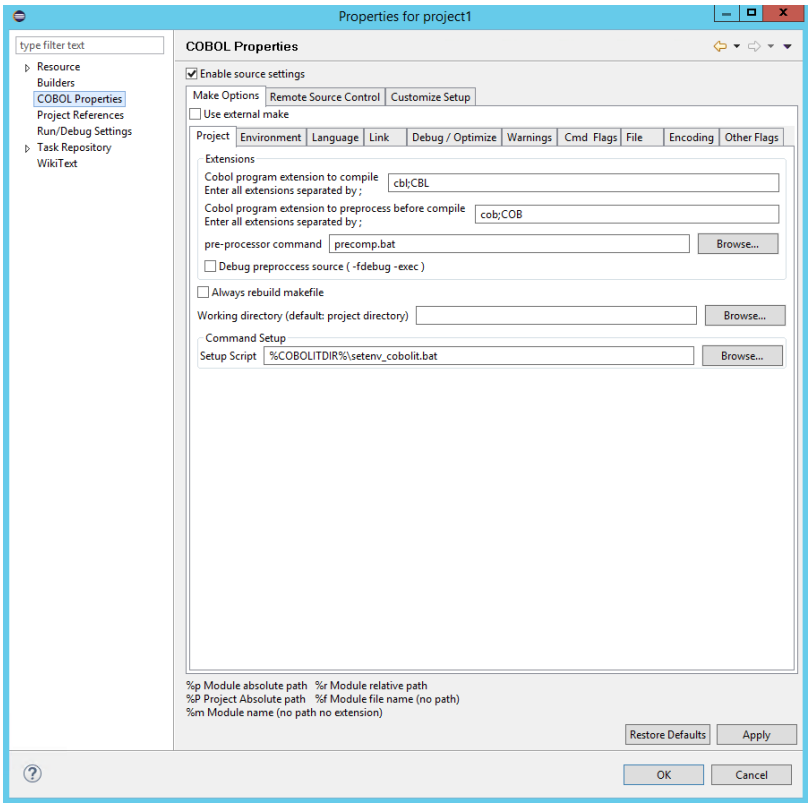
This command logs in to the database server on the local host using port “3 3 0 6” as a user called “test user”, with password “test”, and executes the “show databases” command. If the connection to the database server is successful, you will see that the DBU database exists.

If the user did not exist, or the password did not exist, or the database did not exist, this would be evident, and you would have to correct the situation before proceeding with these exercises.

Project Properties / Project Tab

Set the COBOL program extension to compile to .CBL. Set the COBOL program extension to pre-compile to .COB.

Set pre-processor command to precomp.bat. In our example, precomp.bat is located in the root directory of the project.



Precomp.bat

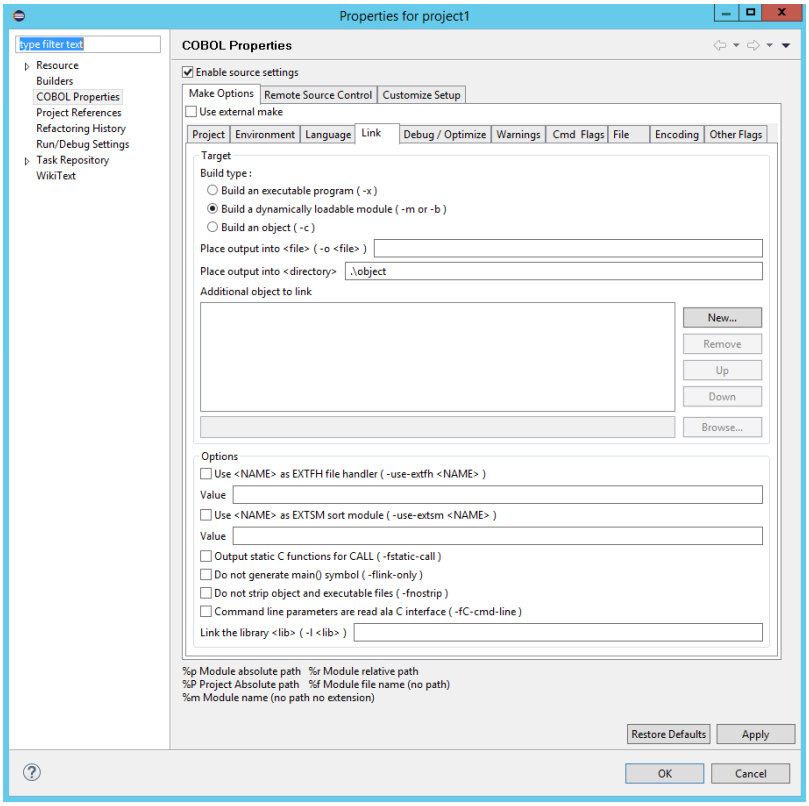
set CITSQLHOME=C:C\COBOL\COBOLIT\distribMYSQL

%CITSQLHOME%\bin\citmysql.exe :IncludeSearchPath=%CITSQLHOME%\include :TargetPattern=%2 %1

Precomp.bat is used by the –preprocess command. In this command, %1 corresponds to the .COB file that you will pre-compile. %2 corresponds to the intermediate file produced by the COBOL-IT Compiler. This file will have a name assigned by the compiler, such as ppiFD1E3148_i.cob.

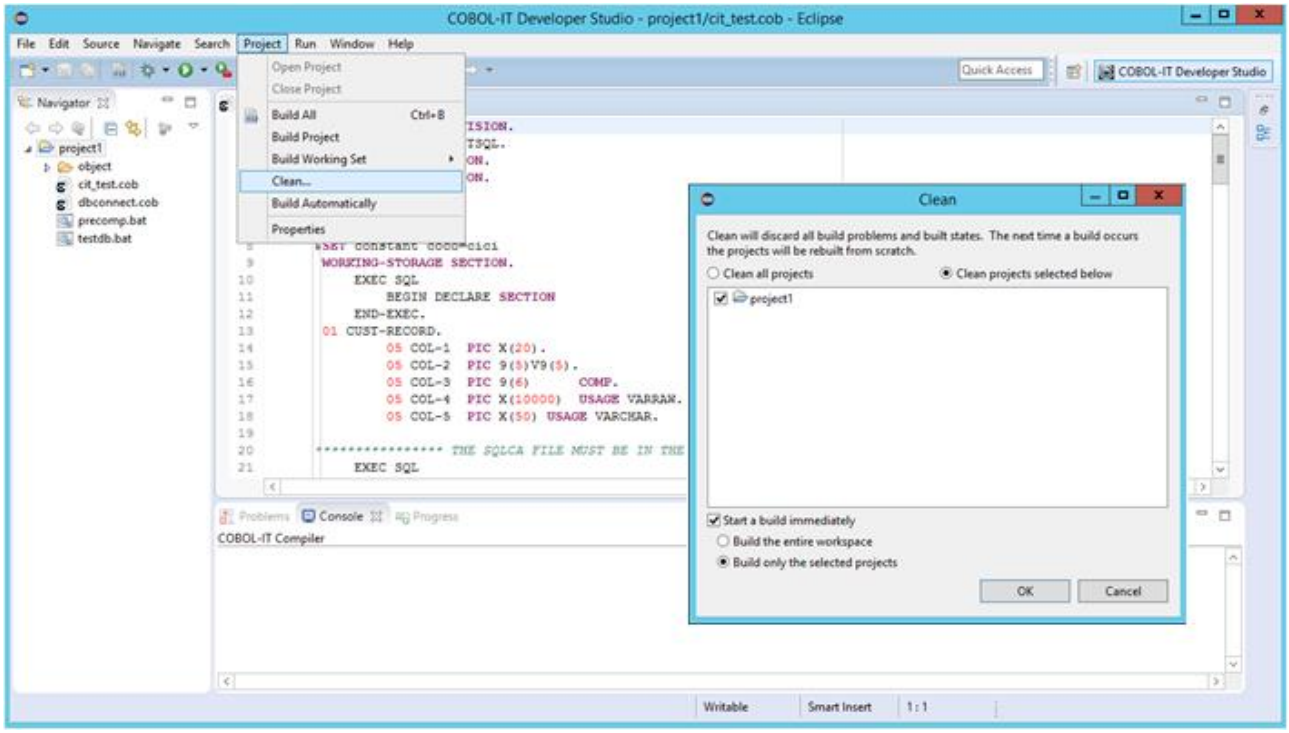
Project Properties / Link Tab

Place the output into a the .\object directory

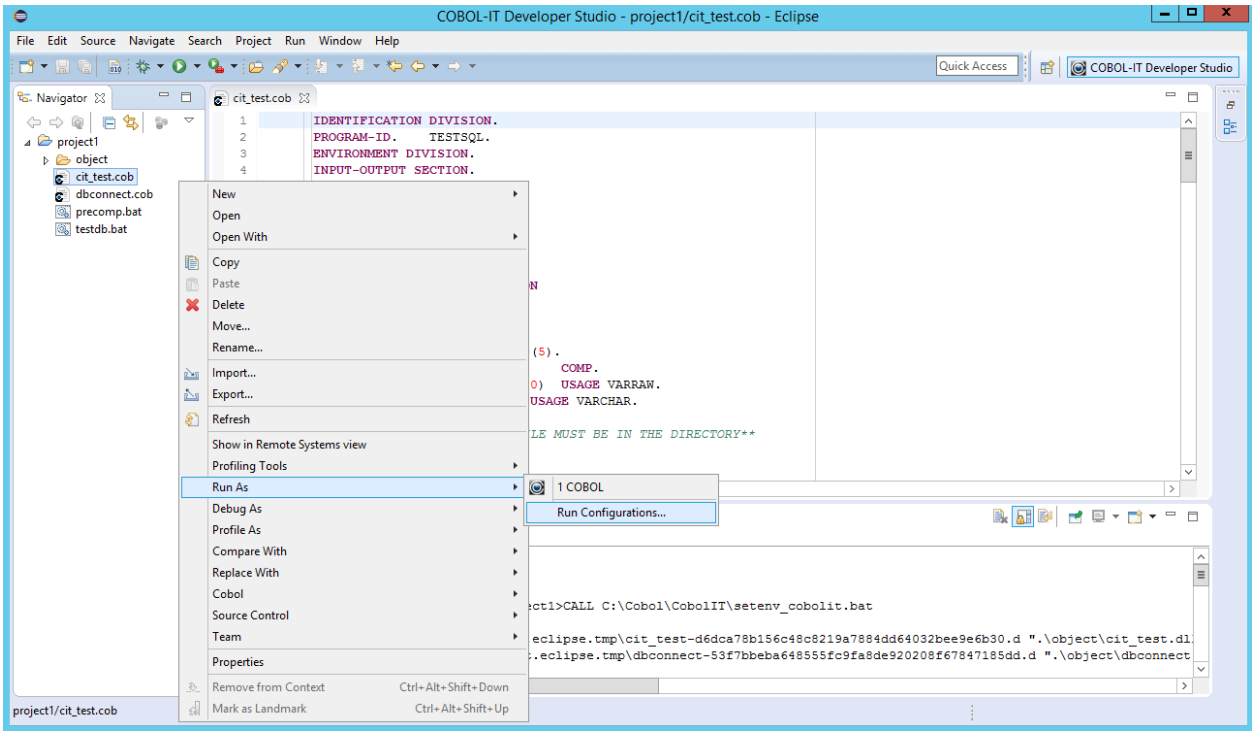


Clean and Build the project

The build will precompile and compile the source file, and generate the compiled objects in the object folder.

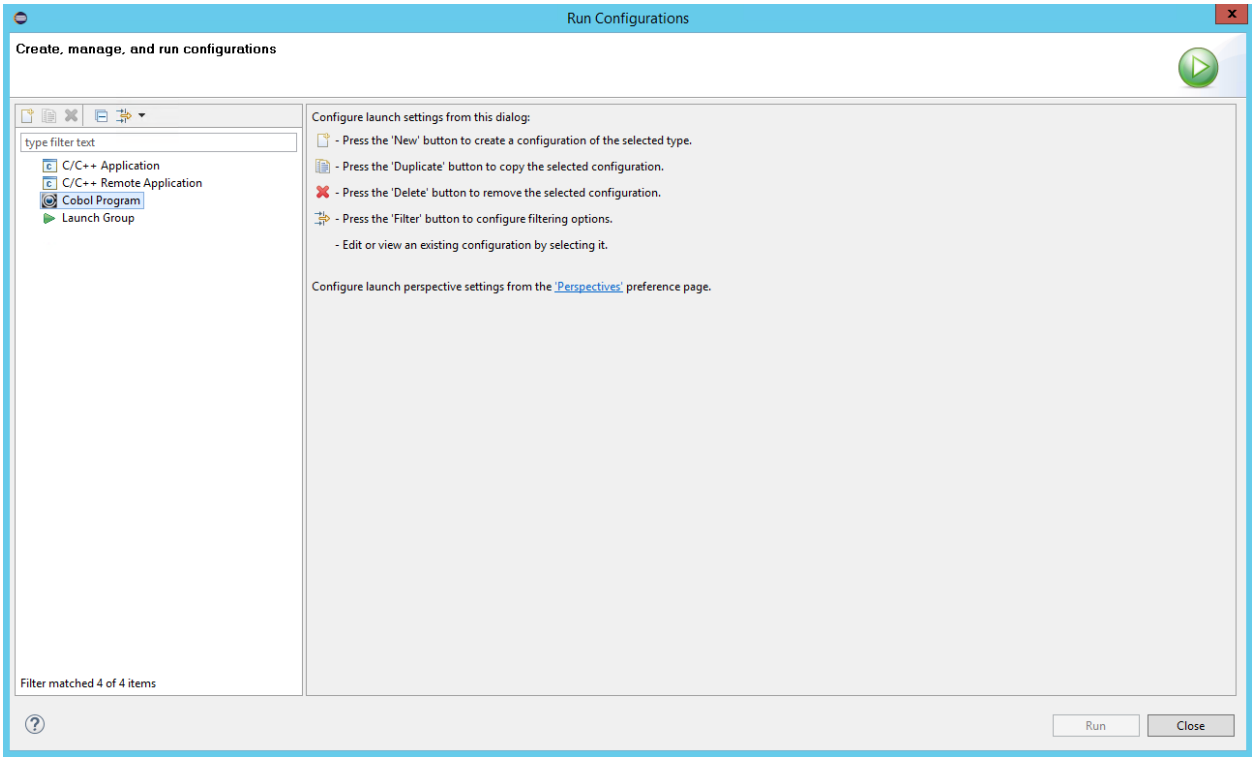


Create a run configuration



The Run Configuration Wizard

Select COBOL Program and Press the New button to create a new configuration.



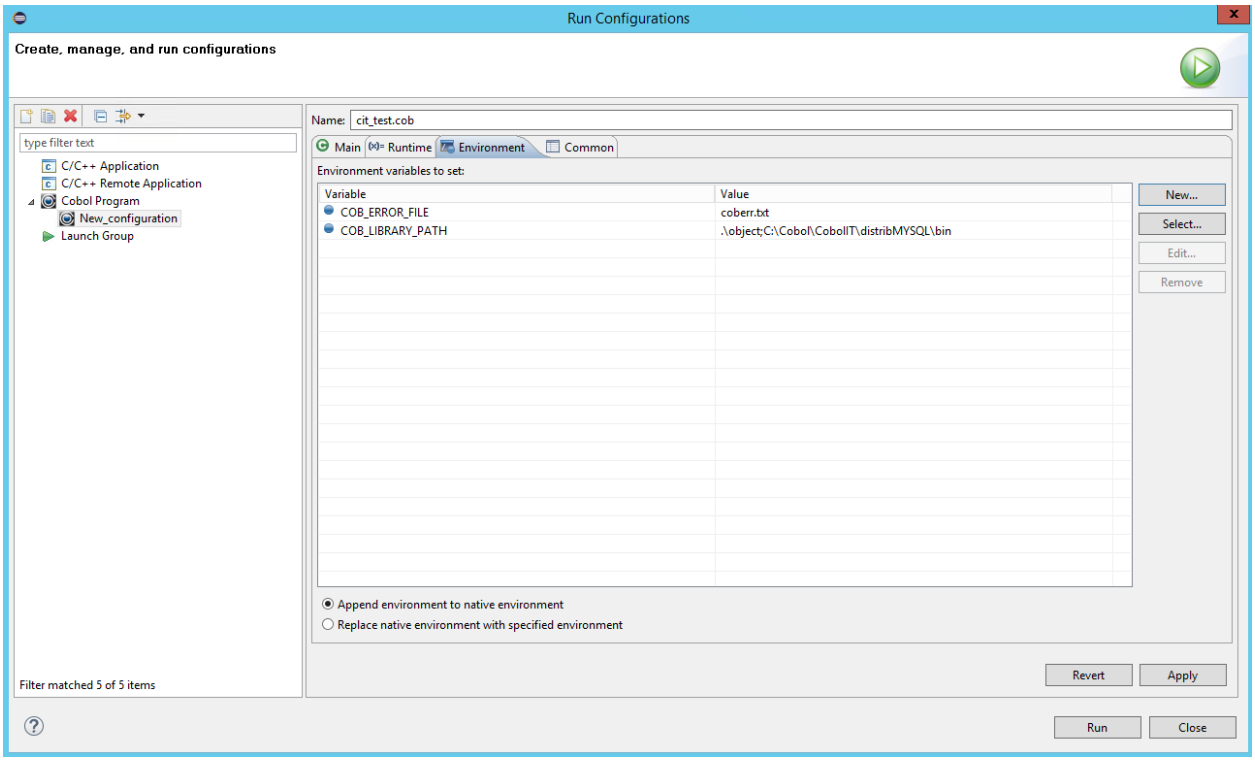
Environment Tab

Use the Defaults for the Main Tab and the Runtime Tab.

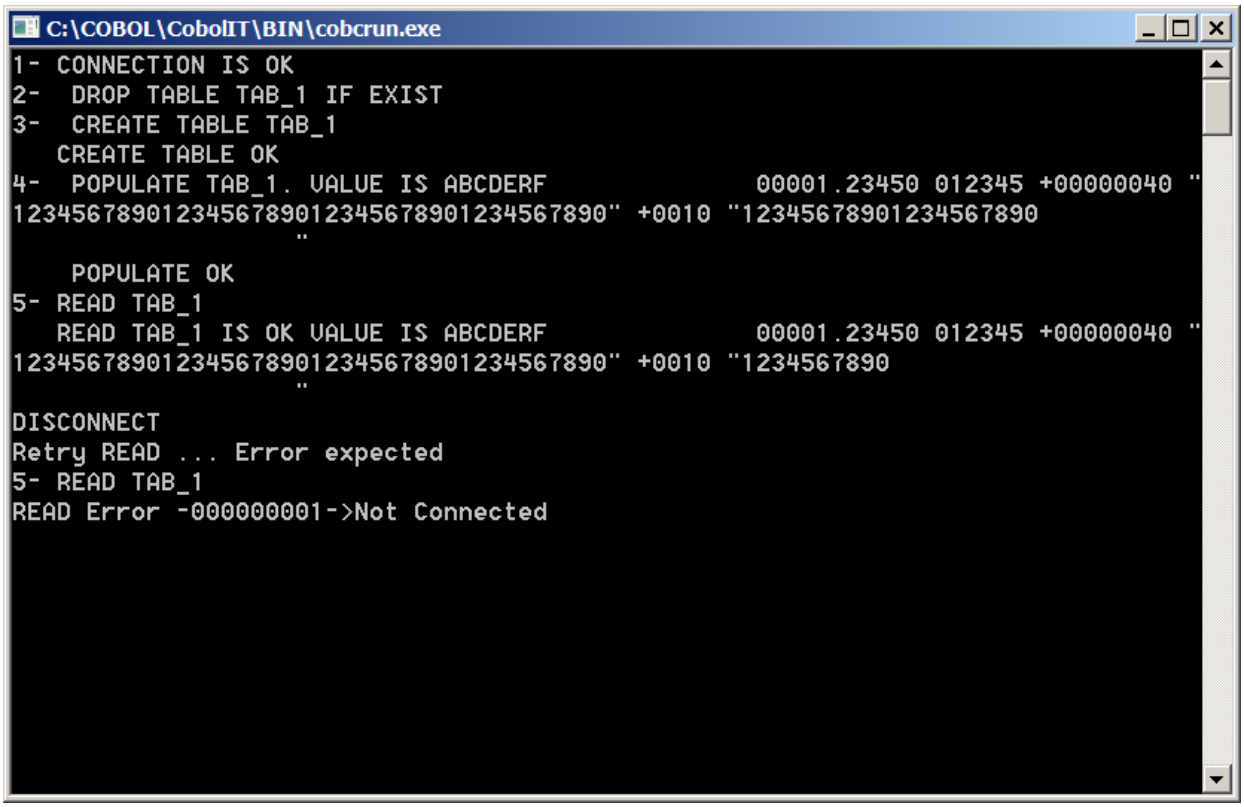
On the Environment Tab:

Set COB_LIBRARY_PATH to .\object; C:\COBOL\COBOLIT\distribMYSQL\bin.

Your object files are in the .\object folder. The RCQMYSQL.DLL file is in the distribMYSQL\bin folder. Click Apply, and the Click the Run button.



The final output



Using CitSQL for PostgreSQL in the Developer Studio

Verify the setup of the database

>psql -d dbu -U testuser -h localhost -w

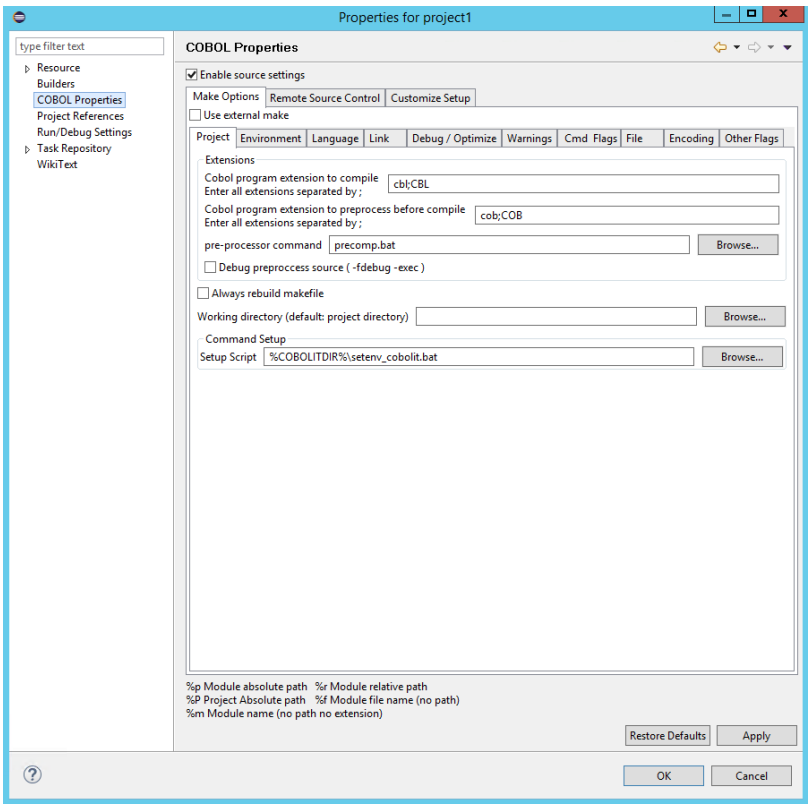
Enter the password, and the command logs in to the database server on the local host as a user called “test user”. If the command is successful, you will see the dbu> prompt.

If the user did not exist, or the password did not exist, or the database did not exist, this would be evident, and you would have to correct the situation before proceeding with these exercises.

Project Properties / Project Tab

Set the COBOL program extension to compile to .CBL. Set the COBOL program extension to pre-compile to .COB.

Set pre-processor command to precomp.bat. In our example, precomp.bat is located in the root directory of the project.



Precomp.bat

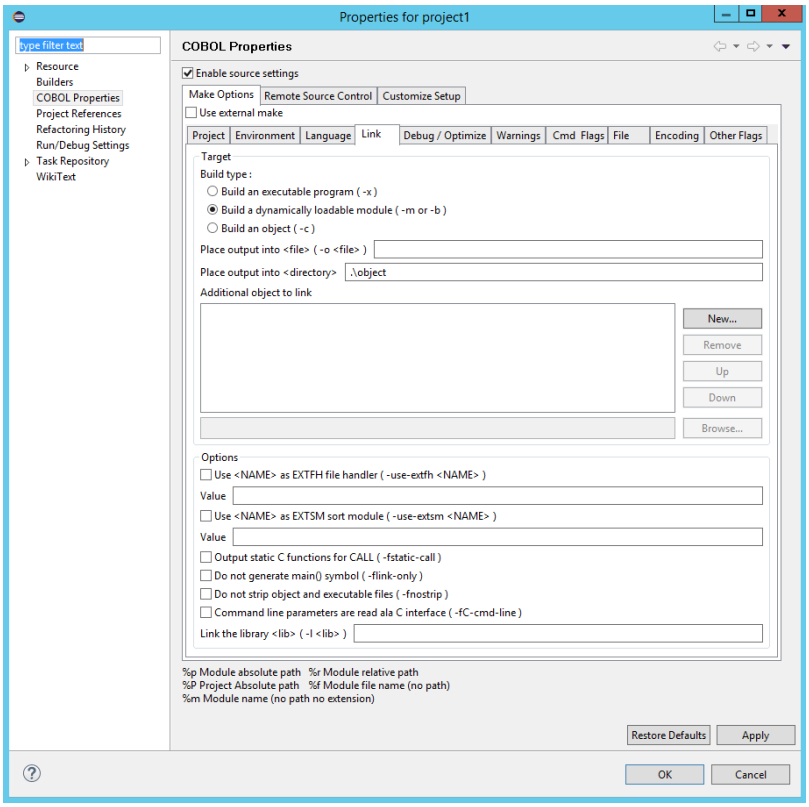
set CITSQLHOME=C:C\COBOL\COBOLIT\distribPGSQL

%CISQLHOME%\bin\citpgsql.exe :IncludeSearchPath=%CITSQLHOME%\include :TargetPattern=%2 %1

Precomp.bat is used by the –preprocess command. In this command, %1 corresponds to the .COB file that you will pre-compile. %2 corresponds to the intermediate file produced by the COBOL-IT Compiler. This file will have a name assigned by the compiler, such as ppiFD1E3148_i.cob.

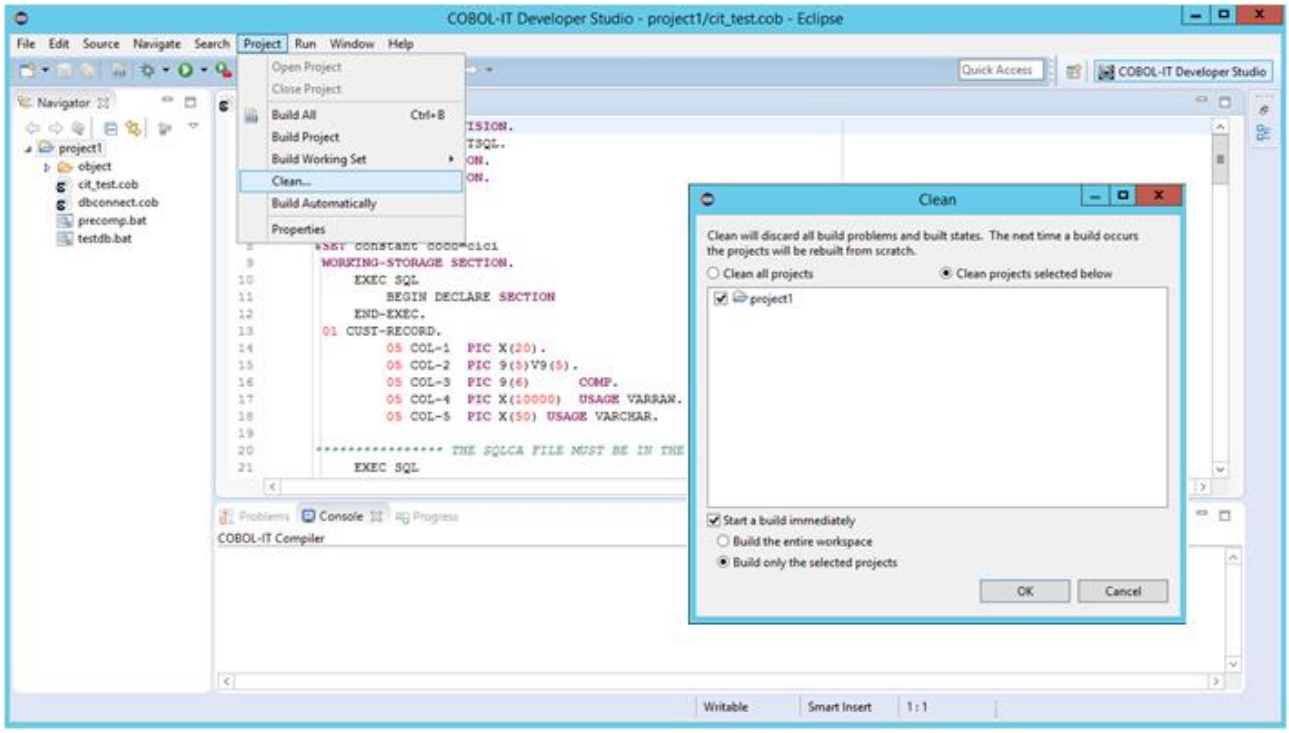
Project Properties / Link Tab

Place the output into a the .\object directory

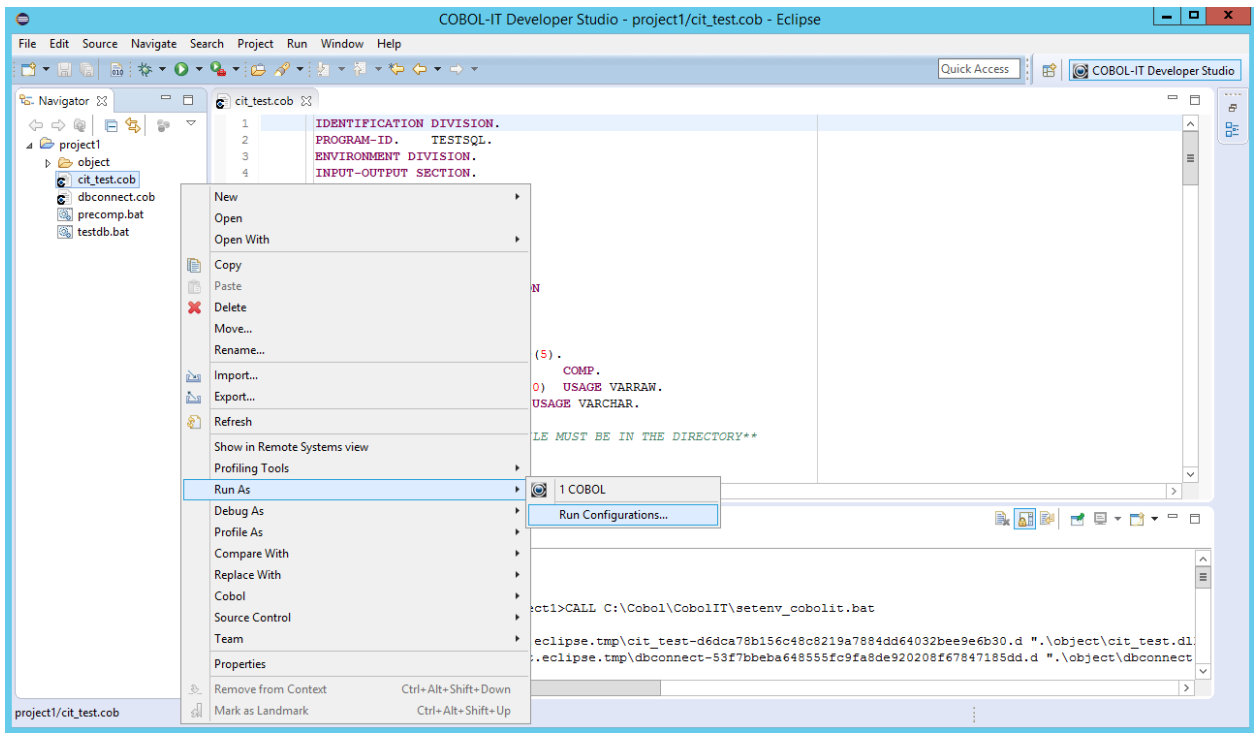


Clean and Build the project

The build will precompile and compile the source file, and generate the compiled objects in the object folder.

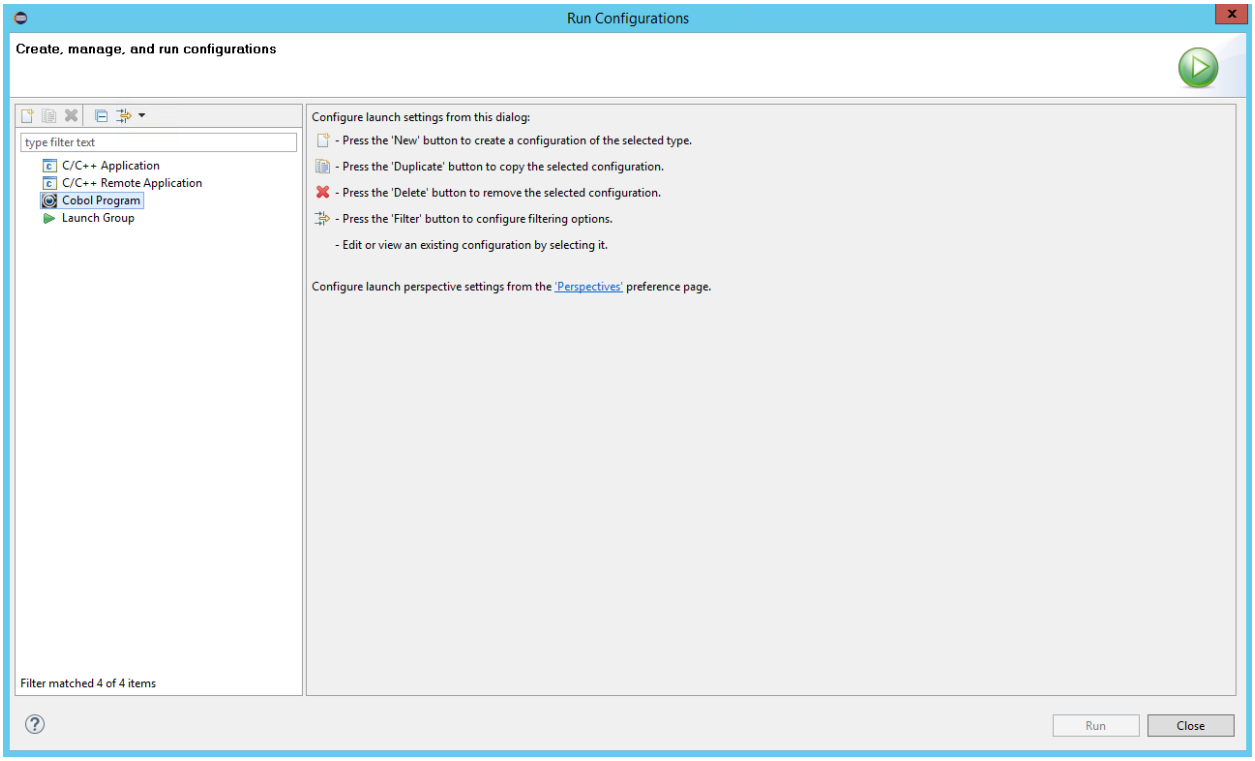


Create a run configuration



The Run Configuration Wizard

Select COBOL Progarm and Press the New button to create a new configuration.



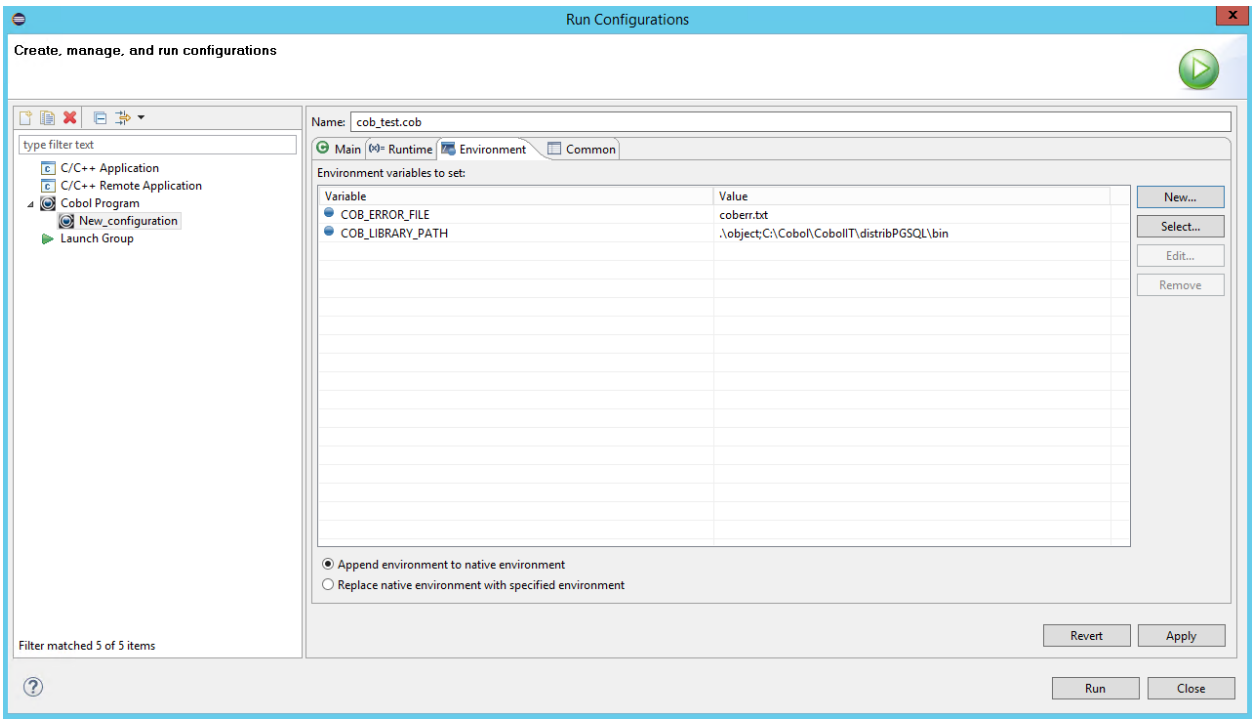
Environment Tab

Use the Defaults for the Main Tab and the Runtime Tab.

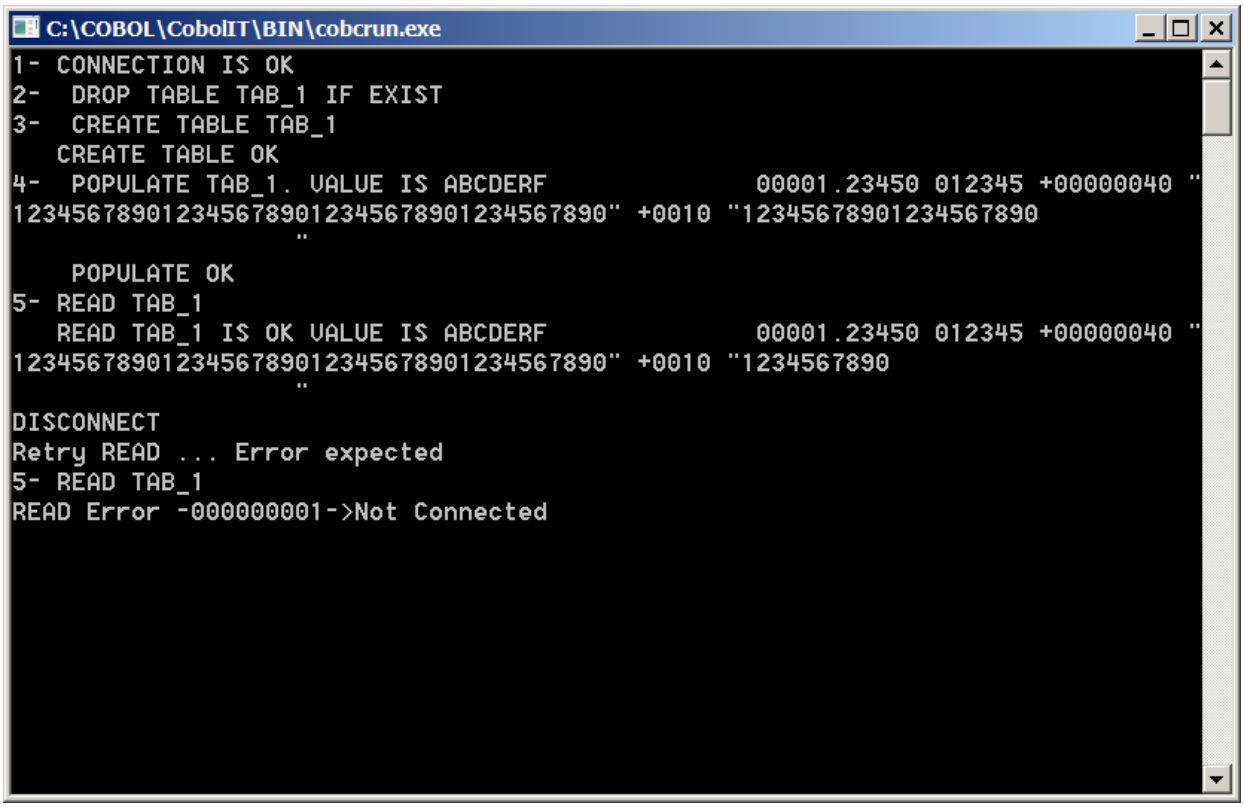
On the Environment Tab:

Set COB_LIBRARY_PATH to .\object; C:\COBOL\COBOLIT\distribPGSQL\bin.

Your object files are in the .\object folder. The RCQPGSQL.DLL file is in the distribPGSQL\bin folder. Click Apply, and the Click the Run button.



The final output

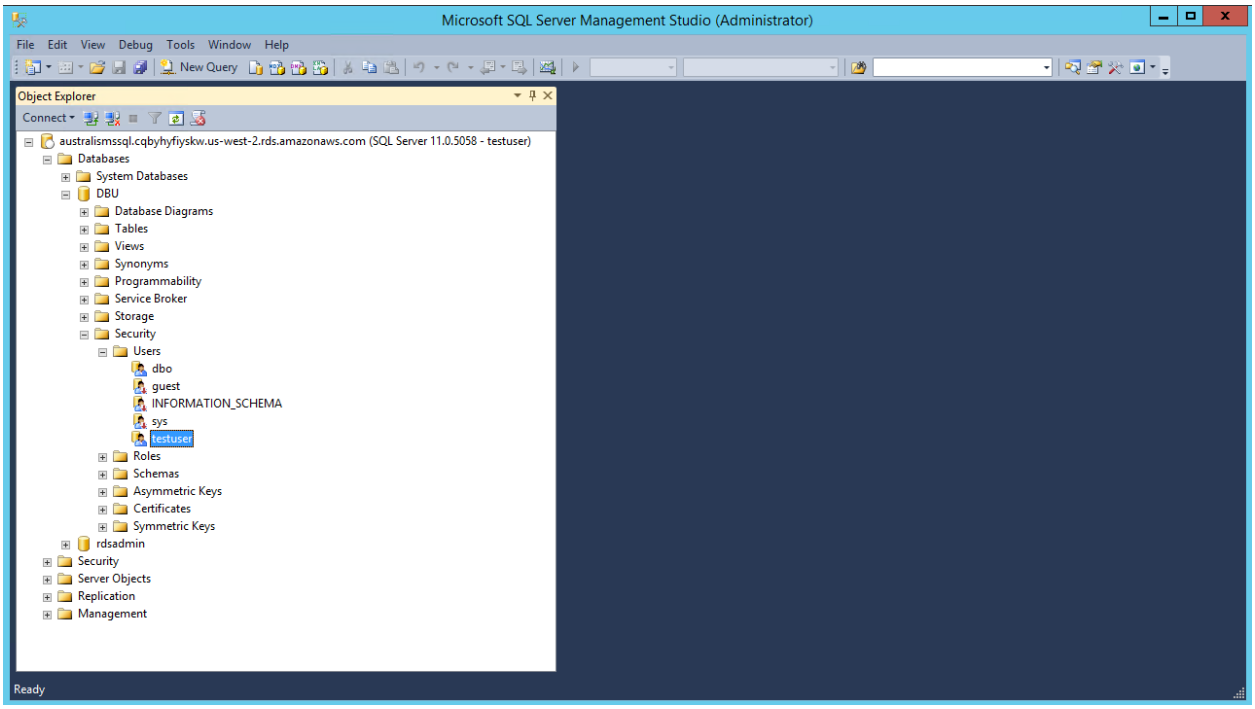


Using CitSQL for ODBC in the Developer Studio

Verify the setup of the database

In the graphic below, in the Microsoft SQL Server Management Studio, we have logged into the database server on the local host as a user called “test user”. The connection to the database server is successful.

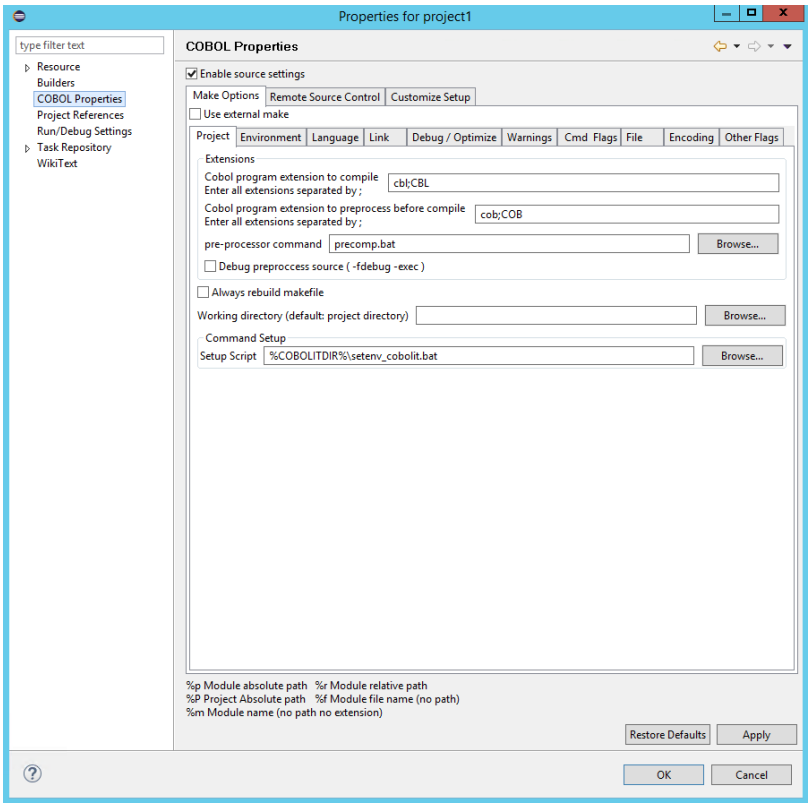
If the user did not exist, or the password did not exist, or the database did not exist, this would have been evident, and you would have to correct the situation before proceeding with these exercises.



Project Properties / Project Tab

Set the COBOL program extension to compile to .CBL. Set the COBOL program extension to pre-compile to .COB.

Set pre-processor command to precomp.bat. In our example, precomp.bat is located in the root directory of the project.



Precomp.bat

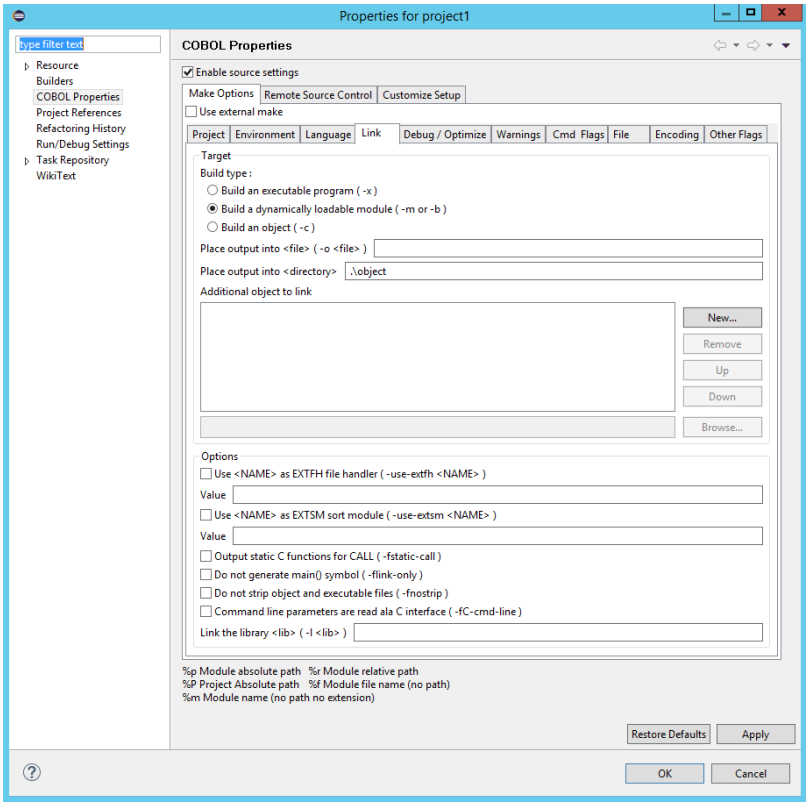
set CITSQLHOME=C:C\COBOL\COBOLIT\distribODBC

%CISQLHOME%\bin\citpgsql.exe :IncludeSearchPath=%CITSQLHOME%\include :TargetPattern=%2 %1

Precomp.bat is used by the –preprocess command. In this command, %1 corresponds to the .COB file that you will pre-compile. %2 corresponds to the intermediate file produced by the COBOL-IT Compiler. This file will have a name assigned by the compiler, such as ppiFD1E3148_i.cob.

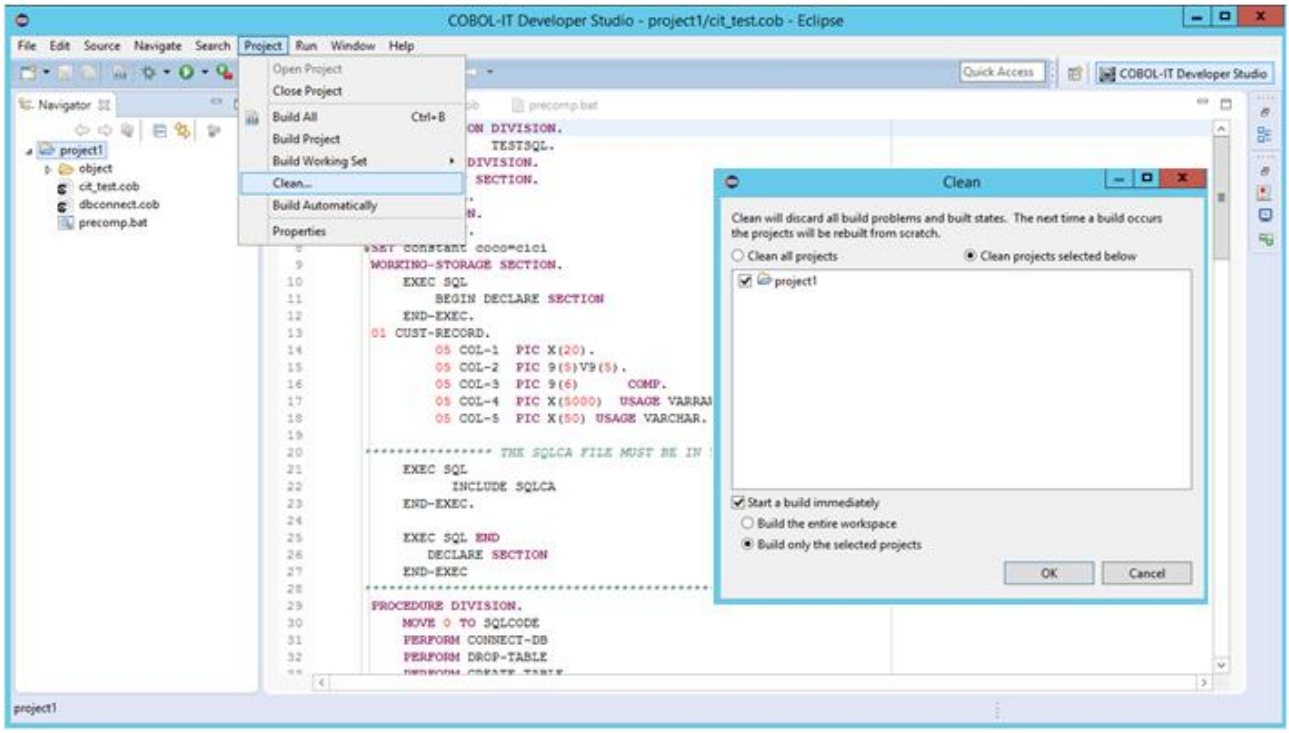
Project Properties / Link Tab

Place the output into a the .\object directory

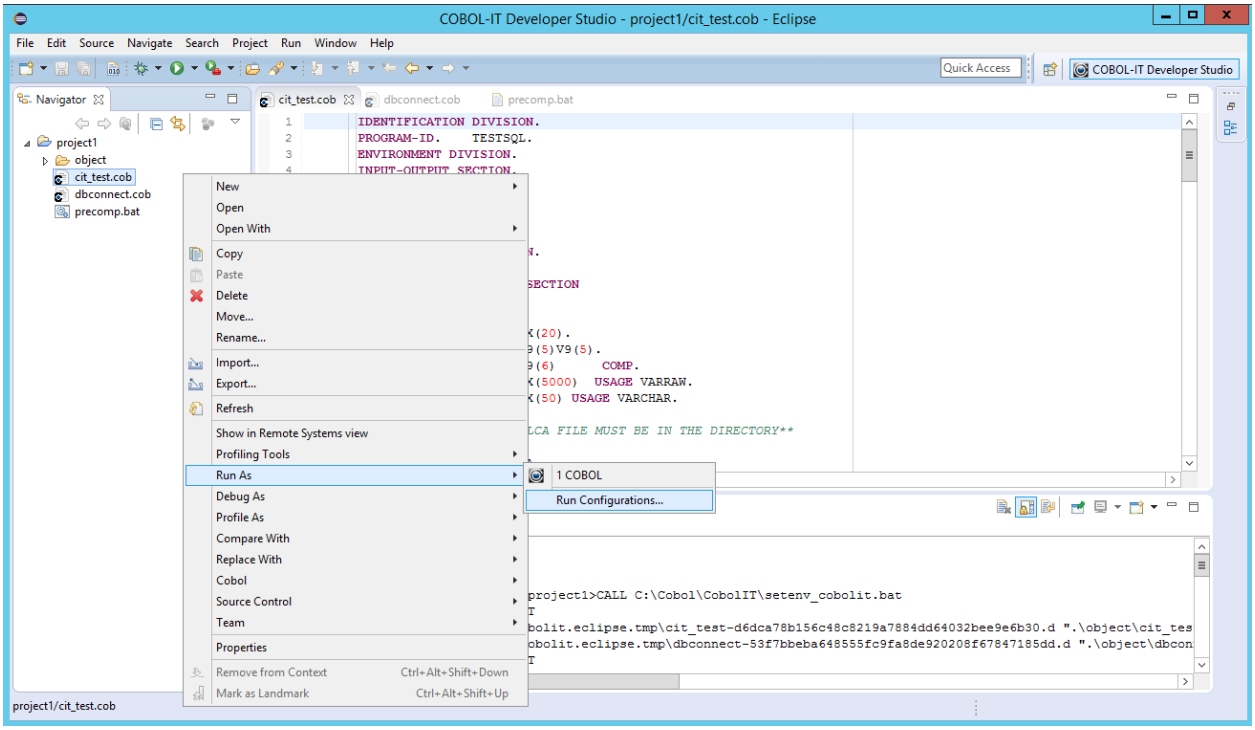


Clean and Build the project

The build will precompile and compile the source file, and generate the compiled objects in the object folder.

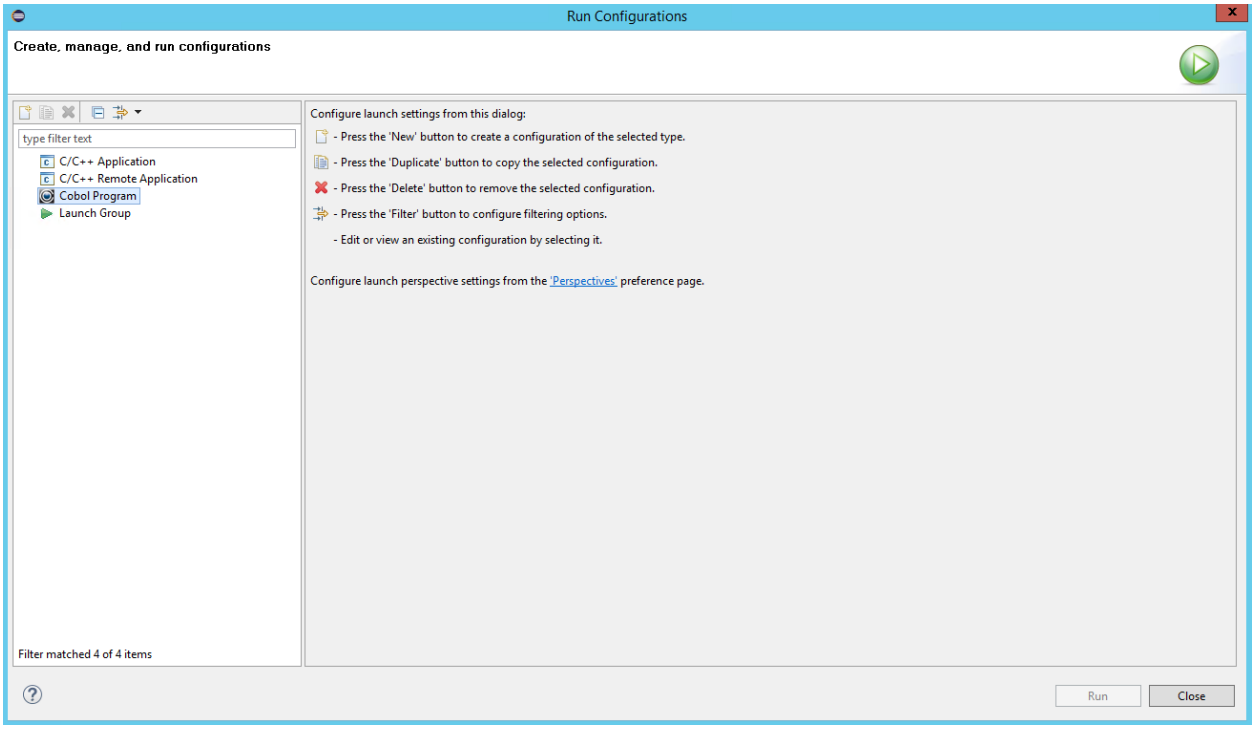


Create a run configuration



The Run Configuration Wizard

Select COBOL Progarm and Press the New button to create a new configuration.



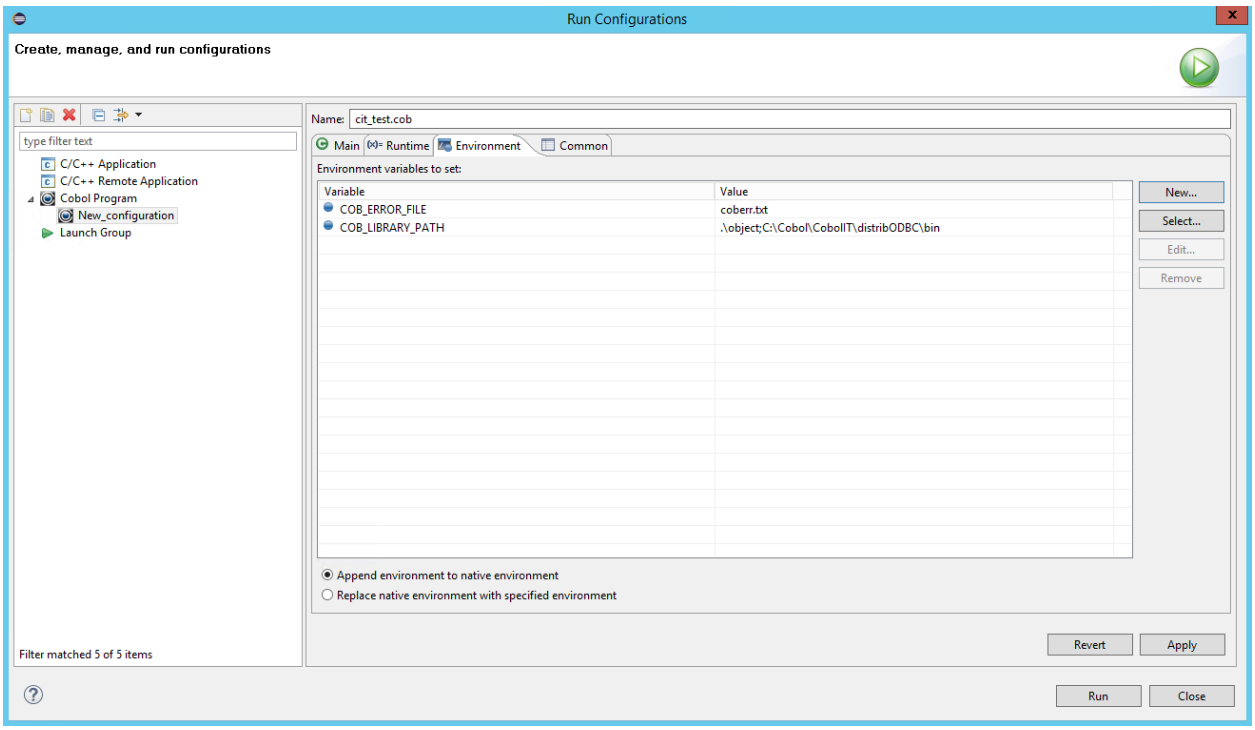
Environment Tab

Use the Defaults for the Main Tab and the Runtime Tab.

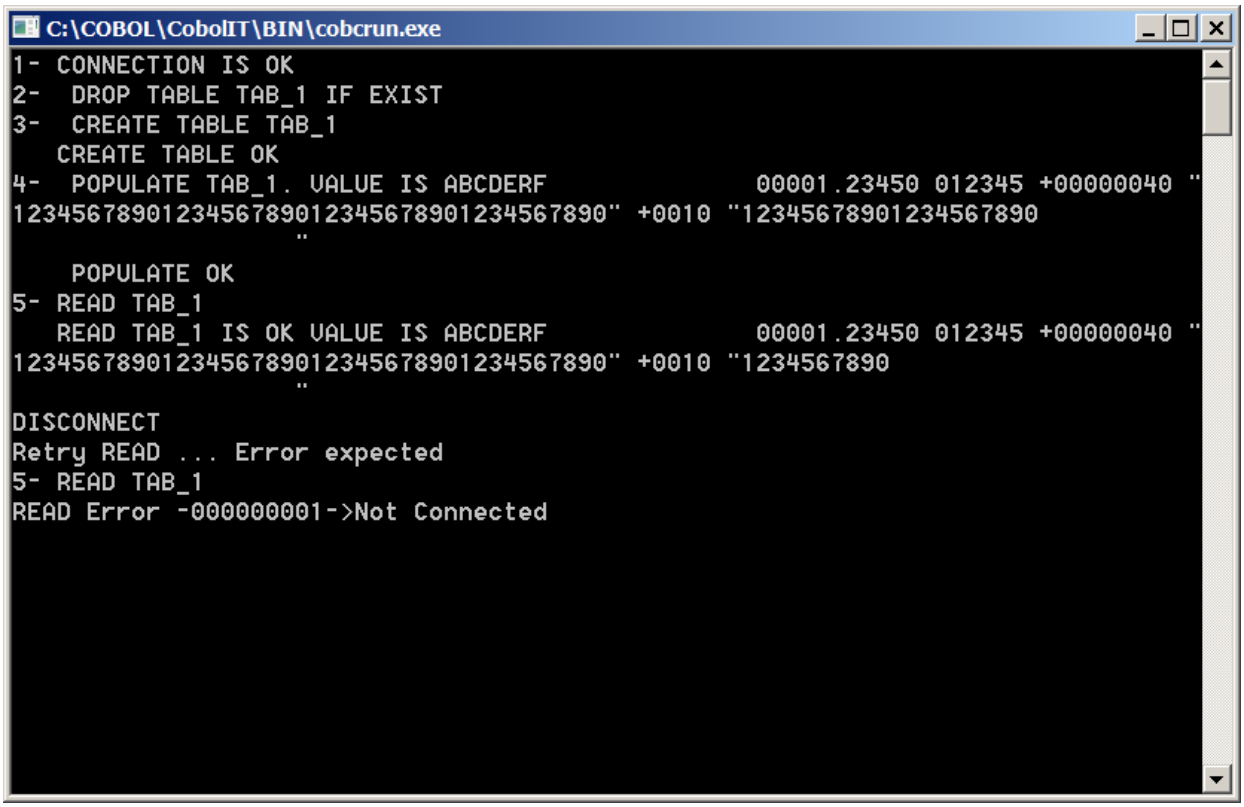
On the Environment Tab:

Set COB_LIBRARY_PATH to .\object; C:\COBOL\COBOLIT\distribODBC\bin.

Your object files are in the .\object folder. The RCODBC.DLL file is in the distribODBC\bin folder. Click Apply, and the Click the Run button.



The final output



Good to know

This section lists details (where the devil usually lies) that can really help you in the process of getting CitSQL to work. This heap of information is not structured, all issues are put together without grouping them in related topics, but one might see this lack of organization as an uncommon form of added value: it forces the reader to consider issues in a totally different area than his/her own. Recommended reading!

By default, the components of a COBOL data structure are concatenated without padding. Even though some COBOL compilers provide options to enable padding (typically to align binary data values to hardware-defined boundaries for optimisation), this facility should *never* be used when compiling precompiled COBOL programs, as the runtime for the precompiler assumes that no padding is used.

A message such as *"Host variable reference in unrecognized block"* indicates that strict mode (See 5.1) is not enforced (as it forbids unrecognized blocks altogether), and that your code contains a SQL construct that is not recognized by CitSQL. When strict mode is not enabled, such SQL statements are sent as is to the database server, as a character string, except when it contains host variable references with a leading colon ("`:XXX-YYY`") as there is no way it can guess whether the host variable should be read or written.

How to get support for CitSQL

In order to get effective support, please note that:

- You need to have a valid ongoing maintenance contract through one of our Subscriptions.
- You must provide detailed information about your operating platform, including operating system, and version .
- You must provide detailed information about the version of the CitSQL precompiler you are using, as well as similarly detailed version regarding the underlying COBOL compiler and target database.
- Identify clearly the problem as being a compile time or a runtime issue.
- Provide a reasonably sized, and possibly, executable, sample so that we can reproduce your problem.

Port Lists

CitSQL for MySQL

Linux x86 32-bits	OS Version	MySQL Version
CitSQL for MySQL for Linux x86 32-bits	Redhat RHEL 4.x	MySQL Version 5 and Later
	Redhat RHEL 5.x	“
	Redhat RHEL 6.x	“
	Redhat RHEL 7.x	“
	SUSE 10.x	“
	SUSE 11.x	“
	OpenSUSE 10.x	“
	Open SUSE 11.x	“
	CENTOS 5.x	“
	CENTOS 6.x	“
	CENTOS 7.x	“
	Oracle Linux 5	“
	Oracle Linux 6	“
	Oracle Linux 7	“
Linux x86 64-bits	OS Version	MySQL Version
CitSQL for MySQL for Linux x86-64 64-bits	Redhat RHEL 4.x	MySQL Version 5 and Later
	Redhat RHEL 5.x	“
	Redhat RHEL 6.x	“
	Redhat RHEL 7.x	“
	SUSE 10.x	“

	SUSE 11.x	“
	OpenSUSE 10.x	“
	Open SUSE 11.x	“
	CENTOS 5.x	“
	CENTOS 6.x	“
	CENTOS 7.x	“
	Oracle Linux 5	“
	Oracle Linux 6	“
	Oracle Linux 7	“
AIX 32-bits	OS Version	MySQL Version
CitSQL for MySQL for AIX 32-bits Xlc	5.x	MySQL Version 5 and Later
	6.x	“
AIX 64-bits	OS Version	MySQL Version
CitSQL for MySQL for AIX 64-bits Xlc	5.x	MySQL Version 5 and Later
	6.x	“
HP-UX Itanium 64-bits	OS Version	MySQL Version
CitSQL for MySQL for HP-UX Itanium 64-bits	11iv2	MySQL Version 5 and Later
	11i v3	“
Sun x86 32-bits	OS Version	MySQL Version
CitSQL for MySQL for Solaris 10 x86 32-bits	Solaris 10	MySQL Version 5 and Later
Sun x86 64-bits	OS Version	MySQL Version
CitSQL for MySQL for Solaris 10 x86 64-bits	Solaris 10	MySQL Version 5 and Later
Sun SPARC 32-bits	OS Version	MySQL Version
CitSQL for MySQL for SUN Solaris 10 SPARC 32-bits	Solaris 10	MySQL Version 5 and Later
Sun SPARC 64-bits	OS Version	MySQL Version
CitSQL for MySQL for SUN Solaris 10 SPARC 64-bits	Solaris 10	MySQL Version 5 and Later
Windows 32-bits	OS Version	MySQL Version
CitSQL for MySQL for Windows 32-bits	Windows 7	MySQL Version 5 and Later
	Windows 8	“
	Windows Server 2012	“
	Windows 10	“

CitSQL for PostgreSQL

Linux x86 32-bits	OS Version	PostgreSQL Version
CitSQL for PostgreSQL for Linux x86 32-bits	Redhat RHEL 4.x	PostgreSQL Version 8 and Later
	Redhat RHEL 5.x	“
	Redhat RHEL 6.x	“
	Redhat RHEL 7.x	“
	SUSE 10.x	“
	SUSE 11.x	“
	OpenSUSE 10.x	“
	OpenSUSE 11.x	“
	CENTOS 5.x	“
	CENTOS 6.x	“
	CENTOS 7.x	“
	Oracle Linux 5	“
	Oracle Linux 6	“
	Oracle Linux 7	“
Linux x86 64-bits	**OS Version**	**PostgreSQL Version**
CitSQL for PostgreSQL for Linux x86-64 64-bits	Redhat RHEL 4.x	PostgreSQL Version 8 and Later
	Redhat RHEL 5.x	“
	Redhat RHEL 6.x	“
	Redhat RHEL 7.x	“
	SUSE 10.x	“
	SUSE 11.x	“
	OpenSUSE 10.x	“
	OpenSUSE 11.x	“
	CENTOS 5.x	“
	CENTOS 6.x	“
	CENTOS 7.x	“
	Oracle Linux 5	“
	Oracle Linux 6	“
	Oracle Linux 7	“
HP-UX Itanium 64-bits	**OS Version**	**PostgreSQL Version**
CitSQL for PostgreSQL for HP-UX Itanium 64-bits	11iv3	PostgreSQL Version 8 and Later
Sun x86 32-bits	**OS Version**	**PostgreSQL Version**
CitSQL for PostgreSQL for Solaris 10 x86 32-bits	Solaris 10	PostgreSQL Version 8 and Later
Sun x86 64-bits	**OS Version**	**PostgreSQL Version**
CitSQL for PostgreSQL for Solaris 10 x86 64-bits	Solaris 10	PostgreSQL Version 8 and Later

Sun SPARC 32-bits	OS Version	PostgreSQL Version
CitSQL for PostgreSQL for SUN Solaris 10 SPARC 32-bits	Solaris 10	PostgreSQL Version 8 and Later
Sun SPARC 64-bits	OS Version	PostgreSQL Version
CitSQL for PostgreSQL for SUN Solaris 10 SPARC 64-bits	Solaris 10	PostgreSQL Version 8 and Later

Windows 32-bits	OS Version	PostgreSQL Version
CitSQL for PostgreSQL for Windows 32-bits	Windows 7	PostgreSQL Version 8 and Later
	Windows 8	"
	Windows Server 2012	"
	Windows 10	"

CitSQL for ODBC

Windows 32-bits	OS Version	ODBC Driver
CitSQL for ODBC for Windows 32-bits	Windows 7	Any ODBC-Compliant Driver
	Windows 8	"
	Windows Server 2012	"
	Windows 10	"

FAQ

Q- I am running the sample program dbconnect on an HP-UX Itanium platform.
After running the program as follows:
>cobcrun dbconnect
I receive the following error:
dbconnect.COB:0: libcob: Cannot find module 'RCQPGSQL'
Please advise.:

A- The precompiler translates EXEC SQL into several CALL "RCQPGSQL"... statements. The RCQPGSQL.so module is provided in /bin directory
You must make it available to the COBOL runtime CALL resolution by placing his path into the COB_LIBRARY_PATH environment variable list.

export COB_LIBRARY_PATH=$COB_LIBRARY_PATH: <location of RCQPGSQL.so>

Q- While executing a compiled program (cobol with sql) , I receive an error. The shared object file **libssl.so.4** is expected, and is not part of your distribution. Please advise.

A- This library is required by libpq.so. COBOL-IT provides a version of libpq.so with the distribution, but it may be different than the version you have installed, so dependencies may be different than expected. In that case, the best course of action is to :

- Install the PostgreSQL libraries from your Linux distribution.
- Rename the lib folder in your CitPostgreSQL distribution.

www.cobol-it.com

June, 2020