# Micro Focus Visual COBOL 2.2J for x64/x86 Linux

# Pro*COBOL による Oracle データベース 12.1c

# アクセス動作検証 検証結果報告書

2015 年 1 月 5 日

マイクロフォーカス株式会社

# 1. 検証概要、目的及びテスト方法

## 1.1 検証概要

Oracle データベースクライアントは、COBOL アプリケーションから Oracle データベースにアクセスするためのプログラミングツールとして Pro*COBOL というプリコンパイラを搭載しています。本ツールを利用しますと、COBOL プログラマは SQL 文を埋め込み形式でプログラムソース中に記述できます。Pro*COBOL プリコンパイラは、COBOL プログラム中に埋め込まれた SQL 文を標準の Oracle ランタイム・ライブラリ・コールに変換します。この Pro*COBOL によるプリコンパイル処理は、Micro Focus の COBOL コンパイラを含めた各種 COBOL コンパイラで翻訳可能な COBOL プログラムを生成します。

Linux 版の Pro*COBOL は本稿執筆時点の段階にて Visual COBOL より 1 つ前の世代の Micro Focus COBOL 開発環境製品シリーズ Micro Focus Server Express5.1 をサポートするとマニュアルに明記しています。

[Oracle Database クイック・インストレーション・ガイド 12c リリース 1 (12.1) for Linux x86-64]
https://docs.oracle.com/cd/E49329_01/install.121/b72980/toc.htm#BABJJFGJ
（2015 年 1 月 5 日リンク確認）

しかしながら、Pro*COBOL によってプリコンパイルされた COBOL プログラムは極めて標準的な COBOL 文法に準拠したものであり、特定のコンパイラバージョンに依存する要因は考えられません。更に、Visual COBOL は稼働保証環境として指定された製品 Server Express 5.1 の後継製品にあたるものであり、コンパイラ機能については上位互換を保証しています。このため Server Express 5.1 向けに展開されるプリコンパイル後の COBOL ソースはそのまま Visual COBOL でも稼働します。

本稿では、Pro*COBOL よりプリコンパイル生成された COBOL プログラムを Visual COBOL でコンパイルし、Oracle データベースへアクセスできることを動作検証しました。また、Visual COBOL に装備された COBSQL 統合プロセッサを利用し、Pro*COBOL によるプリコンパイル、Visual COBOL によるコンパイルをワンステップで処理できることも検証しました。

## 1.2 目的及びテスト方法

Micro Focus Visual COBOL は、Pro*COBOL が稼働保証対象とする Server Express 5.1 のコンパイラ機能に対して下位互換を持つ最新鋭の COBOL 言語開発・実行環境製品です。Oracle は Pro*COBOL 用のサンプルプログラムを提供しています。本検証では

まずそれを使って Pro*COBOL と Visual COBOL の連携により生成されたモジュールが正常に Oracle データベースへアクセスし、データ操作をできることを検証しました。Visual COBOL はコンパイル処理を発行する前に Oracle, Sybase, Informix のプリコンパイラを内部的に呼び出し、ワンステップでプリコンパイル及びコンパイルを処理する COBSQL 統合プロセッサを備えます。Oracle の場合であれば、Pro*COBOL を利用します。本機能を利用すれば、プリコンパイル後の埋め込み SQL 文をライブラリコールに置き換えた COBOL ソースではなく、実際にプログラマがメンテナンスする埋め込み SQL 文が入ったままのロジカルなソースを使って管理できます。Visual COBOL の UNIX/Linux 版には Windows 環境にインストールして使う Eclipse 版の製品がセットになっています。この Eclipse 版製品から UNIX/Linux 環境に接続して開発するリモート開発機能を利用すれば、Eclipse IDE 上で UNIX/Linux 上にあるソースやモジュールに対してエディット、デバッグができます。COBSQL を使う場合、プログラマが実際にメンテナンスする埋め込み SQL 文が入ったプリコンパイル前のソースに対しても Windows ローカルのソースと同様にこれらの開発補助機能を活用して高生産性な開発作業を進めることができます。本稿では Pro*COBOL の検証に使用した埋め込み SQL 文が入ったままの状態のサンプルプログラムを使って COBSQL を利用したリモート開発ができることも検証確認しました。

## 2. 検証環境

- **Database サーバ**

  ソフトウェア

  - Windows 8 Enterprise(VM のゲスト OS として稼働)
  - Oracle Database 12c Release 1(12.1.0.1.0) for Microsoft Windows(x64)
    → Enterprise Edition をインストール
    → Database 作成時に sample schema を作成
  - Oracle Database 12c Release 1 Examples(12.1.0.1.0) for Microsoft Windows(x64)

  ハードウェア

  | | |
  |---|---|
  | 機種： | Dell OPTIPLEX7010 |
  | CPU： | Intel Core i7-3770 3.40GHz |
  | Memory： | 3.00 Gbyte memory(ゲスト OS に割り当てたサイズ) |

- **Database クライアント**

  ソフトウェア

  - Red Hat Enterprise Linux 6 Update 5(VM のゲスト OS として稼働)
  - Oracle Database 12c Release 1 Client(12.1.0.1.0) for Linux x86-64(64bit)
  - Micro Focus Visual COBOL 2.2J Update 2 Development Hub

  ハードウェア

  機種： Dell PowerEdge R7710
  CPU： Intel Xeon CPU X5670 2.93GHz
  Memory： 4.00 Gbyte memory(ゲスト OS に割り当てたサイズ)

- **Visual COBOL クライアント**

  ソフトウェア

  - Windows 8.1 Pro(VM のゲスト OS として稼働)
  - Micro Focus Visual COBOL 2.2J Update 2 for Eclipse
  - Eclipse 3.8(Micro Focus Visual COBOL に同梱されたものを利用)

  ハードウェア

  機種： Dell LATITUDE E6530
  CPU： Intel Core i5-3360 2.80GHz
  Memory： 3.00 Gbyte memory(ゲスト OS に割り当てたサイズ)

## 3. テスト内容

Oracle Database Examples には Pro*COBOL 用のサンプルプログラム及び Micro Focus 製品向けの make ファイルが含まれます。本検証ではこのサンプルプログラムのうち sample4.pco として提供されるプログラムを利用しました。このプログラムには DDL 文、DML 文、DCL 文が含まれこれらの基本的な動作をテストできるようになっています。初めの Pro*COBOL を直接使った検証では、Oracle より提供される同プログラム及び make ファイルを使用してコンパイル・ビルドしました。続く COBSQL を通じた検証においては、make ファイルの内容に合わせてコンパイル及びビルドの命令を構成しました。サンプルに同梱される make ファイルは実行形式へビルドするよう記述されていますが、Micro Focus オリジナルの動的ロードモジュール形式 .gnt へのビルドも検証しています。

# 4. 結果

## 4.1 サンプルアプリケーションの取得

サンプルアプリケーションが格納された Oracle Database 12c Release 1 Examples(12.1.0.1.0) for Linux x86-64 に関しては、Database 製品に追加インストール可能であり、本検証で用意した DB クライアント環境には適用できません。そこで以下の要領で該当のサンプルプログラム及び make ファイルを抽出しました。

① Oracle Database 12c Release 1 Examples(12.1.0.1.0) for Linux x86-64 のインストーラ linuxamd64_12c_examples.zip を取得
② linuxamd64_12c_examples.zip を解凍
③ ＜展開したフォルダ＞¥stage¥Components¥oracle.precomp.companion¥12.1.0.1.0¥1¥DataFiles
配下にある filegroup3.jar を展開
④ 展開したファイルをフォルダ構造を維持したまま Linux Database クライアントの $ORACLE_HOME 配下に格納

## 4.2 サンプルアプリケーションの確認

本検証で利用したサンプルアプリケーションには以下のような処理が含まれます。

① Oracle データベースに接続
② CREATE TABLE 文にてテスト用のテーブルを作成
③ サンプルスキーマに含まれる EMP テーブルよりデータを CURSOR – FETCH で取得
④ 取得したデータ及びプログラム中で加工したデータを INSERT 文にてテスト用のテーブルにデータ充填
⑤ 社員番号の入力を促すプロンプトを出力
⑥ 入力された社員番号をキーに SELECT INTO 文を発行
⑦ 取得したデータを表示
⑧ テスト用のテーブルを DROP
⑨ Oracle データベースとの接続を切断

### 4.3 サンプルアプリケーションの実行結果

サンプルアプリケーションを正常に実行できることを確認しました。詳細は付録の通りとなります。

## 5. テスト結果及び考察

Oracle が提供するプリコンパイラ Pro*COBOL が生成する COBOL プログラムを Visual COBOL を使って正常にコンパイルできることを確認しました。実行形式、動的ロードモジュール形式問わずコンパイルしたアプリケーションが正常に動作することも確認しています。同アプリケーションが正常に処理されたことにより代表的な DDL 文、DML 文、DCL 文を Pro*COBOL と Visual COBOL の組み合わせで問題なく扱えることが検証できました。また、COBSQL を使ってシングルステップで同アプリケーションを開発し、埋め込み SQL 文が入ったままのソースで管理できることも確認しました。つまりこのプリコンパイル前のソースに対して Visual COBOL が COBOL 開発用に Eclipse IDE に作りこんだ開発補助機能を駆使して、効率的に Oracle データベースと連携する COBOL アプリケーションの開発ができることを確認できています。

以上

## 付録１． Linux – Pro*COBOL

1) プログラムの先頭行にてコンパイラオプションを上書き

```
$ head -1 sample4.pco
      $SET CURRENCY-SIGN(36)
$
```

日本語ロケール配下で製品インストールした場合、CURRECNY-SIGN コンパイラオプションにはデフォルトでは「¥(92)」が指定されるよう構成されます。Oracle から提供されるサンプルプログラムでは通貨記号に「$(36)」を使用しているため、コンパイラに「$」を通貨記号として認識させてあげるべくオプションを上書きます。

2) プログラム中の接続処理部分を検証環境に合わせて書き換え

編集前：

```
$ cat sample4.pco
                                    :
       01  USERNAME          PIC X(10) VARYING.
       01  PASSWD            PIC X(10) VARYING.
                                    :
           MOVE "scott" TO USERNAME-ARR.
           MOVE 5 TO USERNAME-LEN.
           MOVE "tiger" TO PASSWD-ARR.
           MOVE 5 TO PASSWD-LEN.
           EXEC SQL
               CONNECT :USERNAME IDENTIFIED BY :PASSWD
           END-EXEC.
                                    :
$
```

編集後

```
$ cat sample4.pco
                                    :
       01  CONNSTR           PIC X(20) VARYING.
                                    :
           MOVE "scott/tiger@orcl" TO CONNSTR-ARR.
           MOVE 16 TO CONNSTR-LEN.
           EXEC SQL
               CONNECT :CONNSTR
           END-EXEC.
                                    :
$
```

3) Pro*COBOL のサンプル中に含まれる make ファイルを使って、サンプルプログラムをプリコンパイル及びコンパイル

```
$ make -f demo_procob.mk sample4
make -f /home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/precomp/demo/procob2/demo_procob.mk build COBS=sample4.cob EXE=sample4
make[1]: ディレクトリ `/home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/precomp/demo/procob2' に入ります
procob  iname=sample4.pco

Pro*COBOL: Release 12.1.0.1.0 - Production on 月 12 月 22 10:35:00 2014

Copyright (c) 1982, 2013, Oracle and/or its affiliates.  All rights reserved.

システムのデフォルト・オプション値: /home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/precomp/admin/pcbcfg.cfg

cob -C IBMCOMP -C NESTCALL -x -t -o sample4 sample4.cob -L/home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/lib/ /home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/precomp/lib/cobsqlintf.o -lclntsh -lclntshcore `cat /home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/lib/ldflags`  `cat /home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/lib/sysliblist` -ldl -lm
* 無視  - NESTCALL
make[1]: ディレクトリ `/home/yoshihiro/app/yoshihiro/product/12.1.0/client_1/precomp/demo/procob2' から出ます
$
```

※1

※2

※1: Pro*COBOL によるプリコンパイル。sample4.pco から sample4.cob を生成。

※2: Visual COBOL のコンパイルコマンドを使って sample4.cob をコンパイル。

4) プログラムを実行

```
$ ./sample4

CONNECTED TO ORACLE VIA STMT:  scott/tiger@orcl

OK TO DROP THE IMAGE TABLE? (Y/N)  Y

TABLE IMAGE DOES NOT EXIST - CREATING NEW TABLE.

INSERTING BITMAPS INTO IMAGE FOR ALL EMPLOYEES ...

EMPLOYEE SMITH     ..............................  IS DONE!
EMPLOYEE ALLEN     ..............................  IS DONE!
EMPLOYEE WARD      ..............................  IS DONE!
EMPLOYEE JONES     ..............................  IS DONE!
EMPLOYEE MARTIN    ..............................  IS DONE!
EMPLOYEE BLAKE     ..............................  IS DONE!
EMPLOYEE CLARK     ..............................  IS DONE!
EMPLOYEE KING      ..............................  IS DONE!
```

```
EMPLOYEE TURNER    ............................... IS DONE!
EMPLOYEE JAMES     ............................... IS DONE!
EMPLOYEE FORD      ............................... IS DONE!
EMPLOYEE MILLER    ............................... IS DONE!

DONE INSERTING BITMAPS.  NEXT, LET'S DISPLAY SOME.

ENTER EMPLOYEE NUMBER (0 TO QUIT):  7654

               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************

EMPLOYEE MARTIN     HAS SALARY $ 1250.00 AND COMMISSION $ 1400.00.

ENTER EMPLOYEE NUMBER (0 TO QUIT):  7900

               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************
               **************************

EMPLOYEE JAMES      HAS SALARY $  950.00 AND NO COMMISSION.

ENTER EMPLOYEE NUMBER (0 TO QUIT):  0

HAVE A GOOD DAY.

$
```

➔ 全て正常に処理できていることが確認できます。

## 付録２． Linux – COBSQL(実行形式へのビルド)

1) Oracle のライブラリをリンクインするためのオプションファイルを生成[1]

```
$ $COBDIR/src/oracle/set_cobopt_oracle
Set COBOPT to /home/yoshihiro/cobopt.ora before starting the RDO daemon.
Ensure that you specify both the main entry point name and the
output name when linking your user application.
From the command-line, you can do this by passing
entry_point -o output_name to cob.
$
```

2) Root 権限を持ったユーザでログイン

3) Visual COBOL 用の環境設定スクリプトを実行後、1) で生成したオプションファイルを環境変数 COBOPT でポイント

```
# export COBOPT=`pwd`/cobopt.ora
#
```

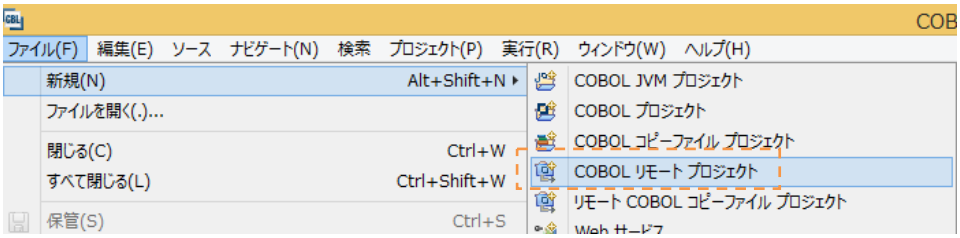
4) リモート開発用のデーモンを起動

```
# $COBDIR/remotedev/startrdodaemon
Checking Java Version
Correct Java Version installed, proceeding
Starting RSE daemon...
Daemon running on: tok-rhel65-64, port: 4075
                                    :
```

[1] Pro*COBOL により生成された COBOL プログラムを実行形式のモジュールにビルドするには付録１ 3) に表示されているように Oracle が提供するライブラリをリンクリンする必要があります。$COBDIR/src/oracle/set_cobopt_oracle スクリプトを実行するとカレントディレクトリの cobopt.ora にこのリンクインオプションを書き出します。

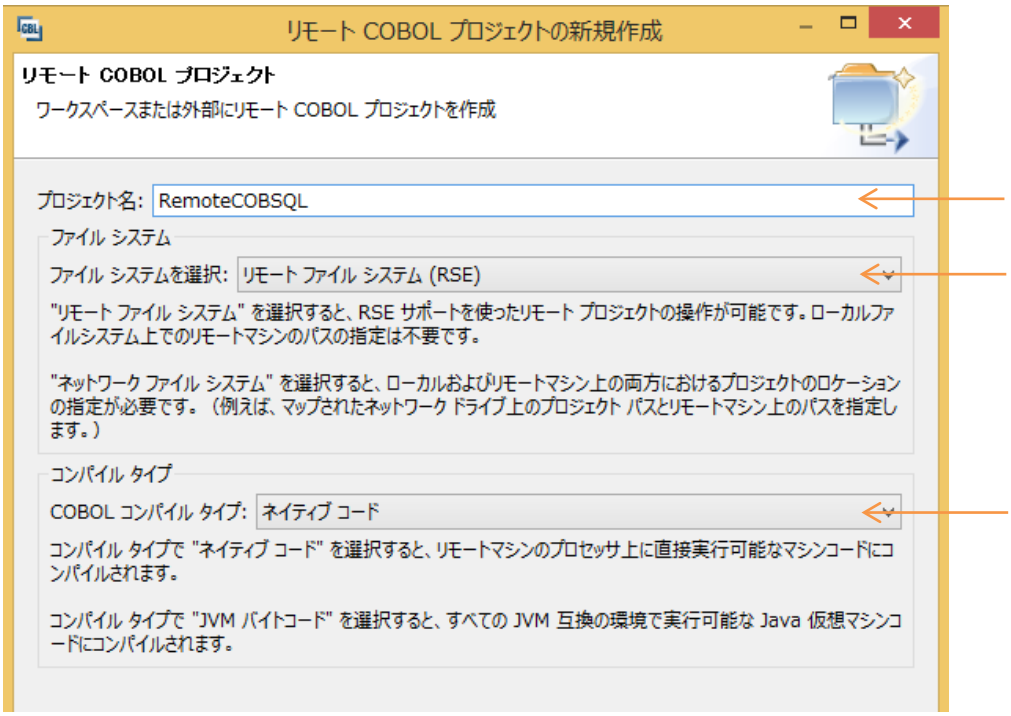
5) Windows 環境にて Visual COBOL for Eclipse を起動

6) [ファイル]メニュー > [新規(N)] > [COBOL リモートプロジェクト]を選択し COBOL リモートプロジェクトを作成

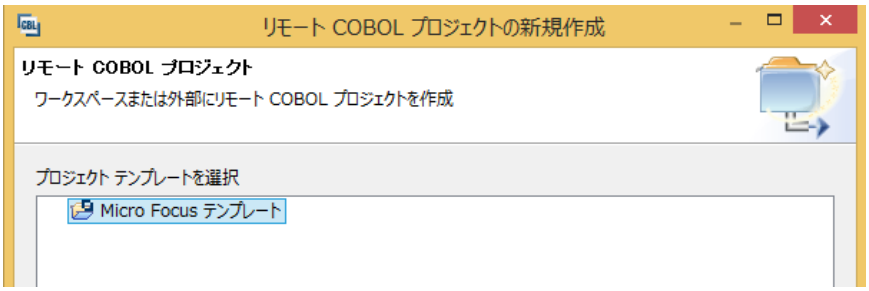



7) プロジェクトの作成ウィザードの１つ目の画面にて以下の情報を指定

プロジェクト名　・・・　任意のプロジェクト名
ファイルシステム　・・・　リモートファイルシステム(RSE)
コンパイルタイプ　・・・　ネイティブコード

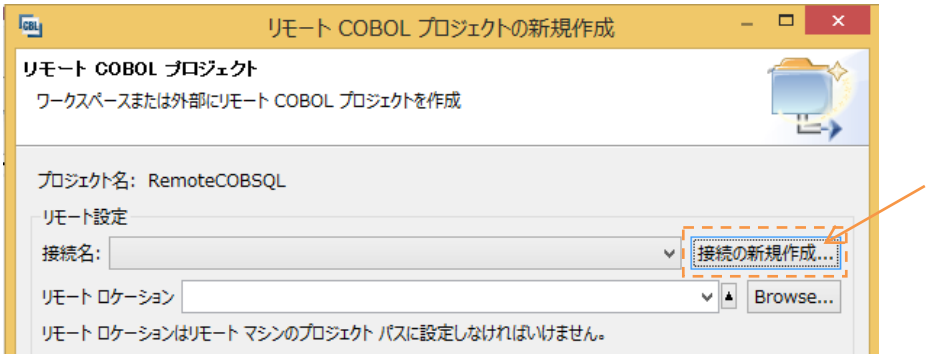


8) プロジェクトの作成ウィザードの２つ目の画面ではデフォルトの［Micro Focus テンプレート］を選択し［次へ］ボタンを押下

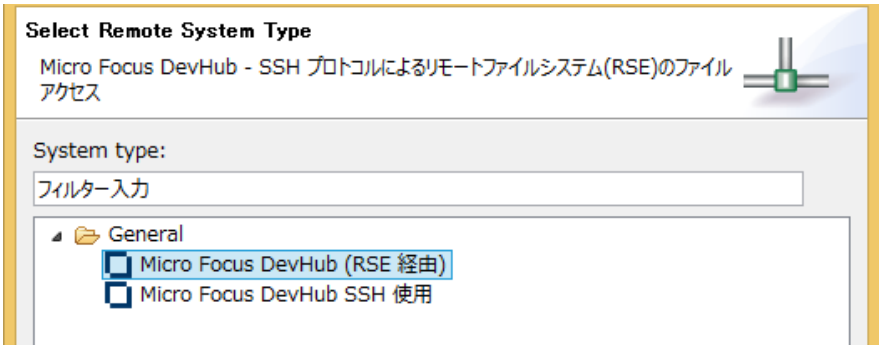



9) 続く画面にて Linux 側への接続情報を指定

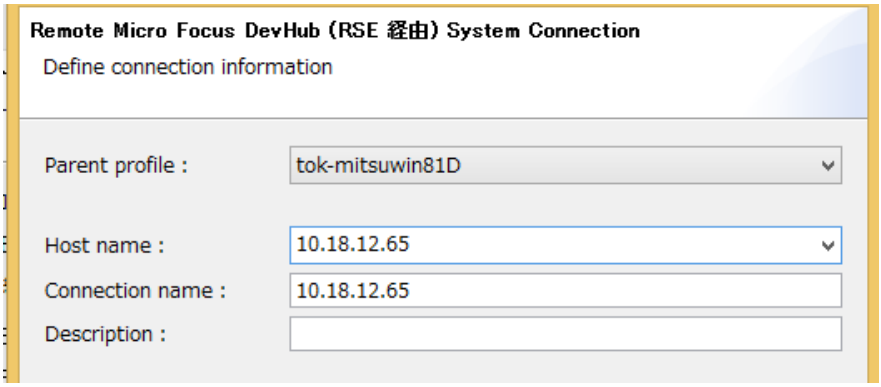
① ［接続の新規作成］ボタンを押下




② 「Micro Focus DevHub(RSE 経由)」を選択して［次へ］ボタンを押下

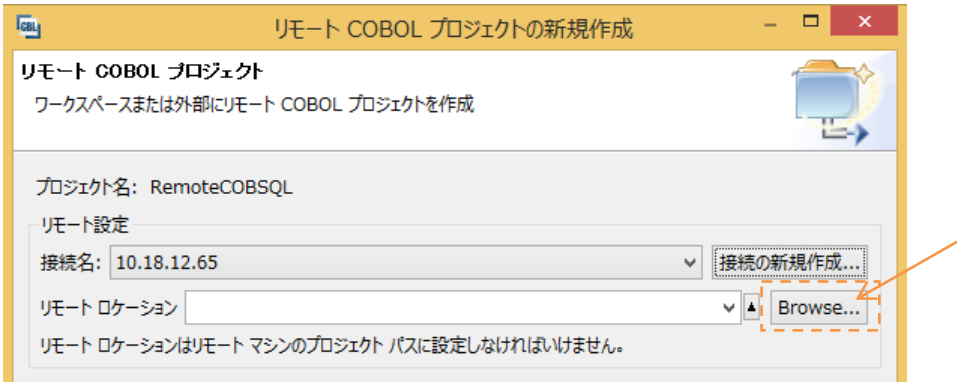


③　ホスト名もしくは IP アドレスを入力して［完了］ボタンを押下]

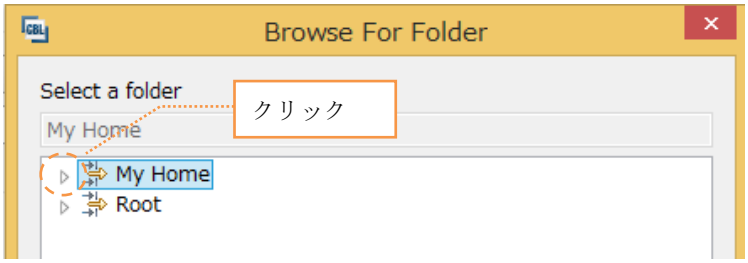



10) Linux 側に配備する Eclipse のプロジェクトディレクトリを指定

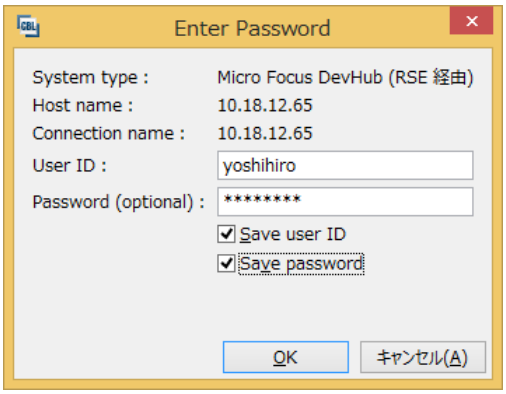
①　［リモートロケーション］欄にて［Browse］ボタンを押下



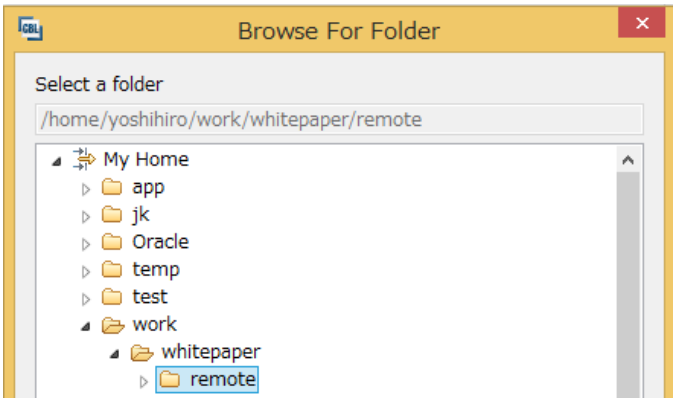

②　「My Home」を展開



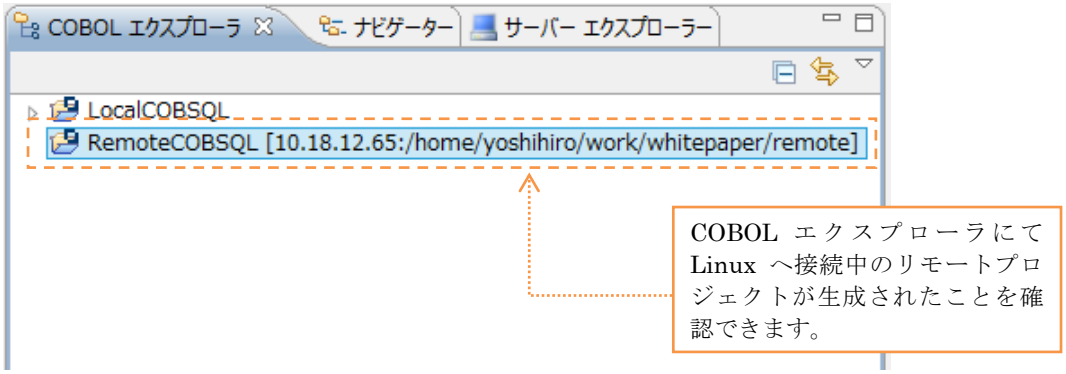
③　Linux 側のユーザログイン情報を指定して［OK］ボタンを押下




④　プロジェクトディレクトリとして利用するディレクトリを選択して［OK］ボタンを押下

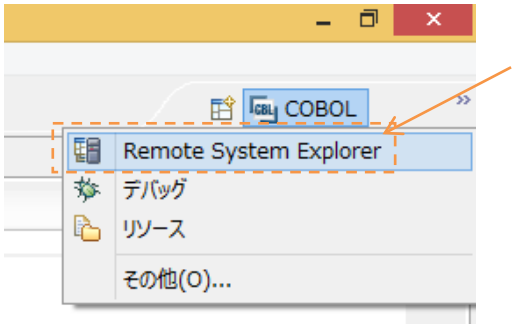



⑤　［完了］ボタンを押下

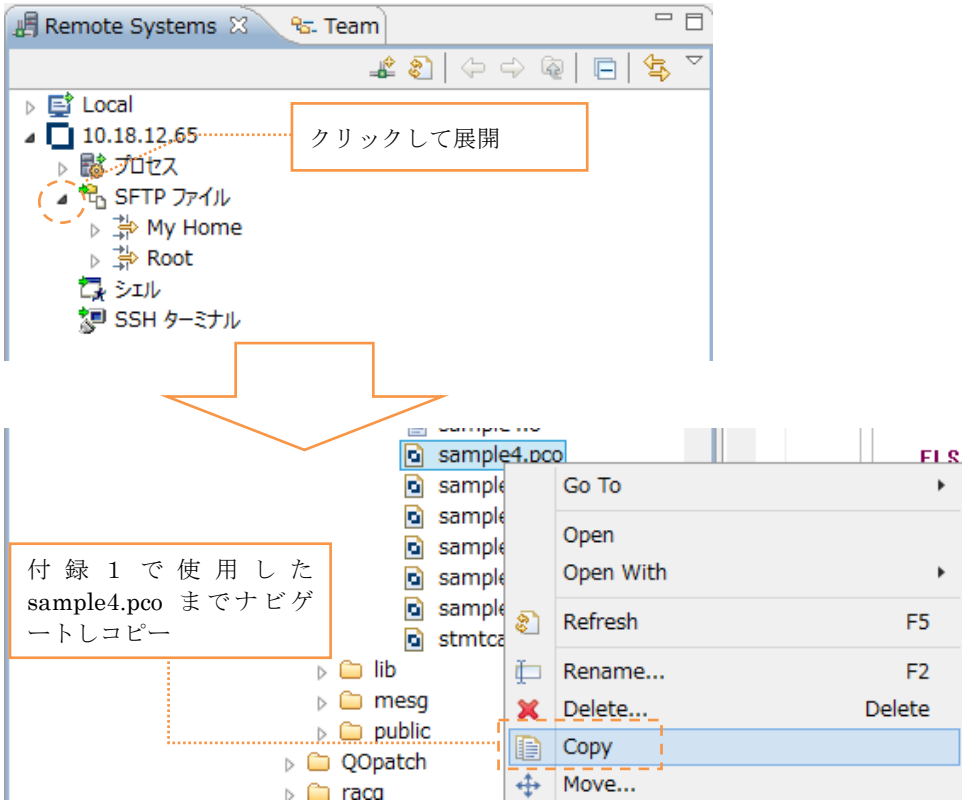


11) リモート開発用に作成したプロジェクトディレクトリに付録 1 で使用したサンプルプログラムへコピー

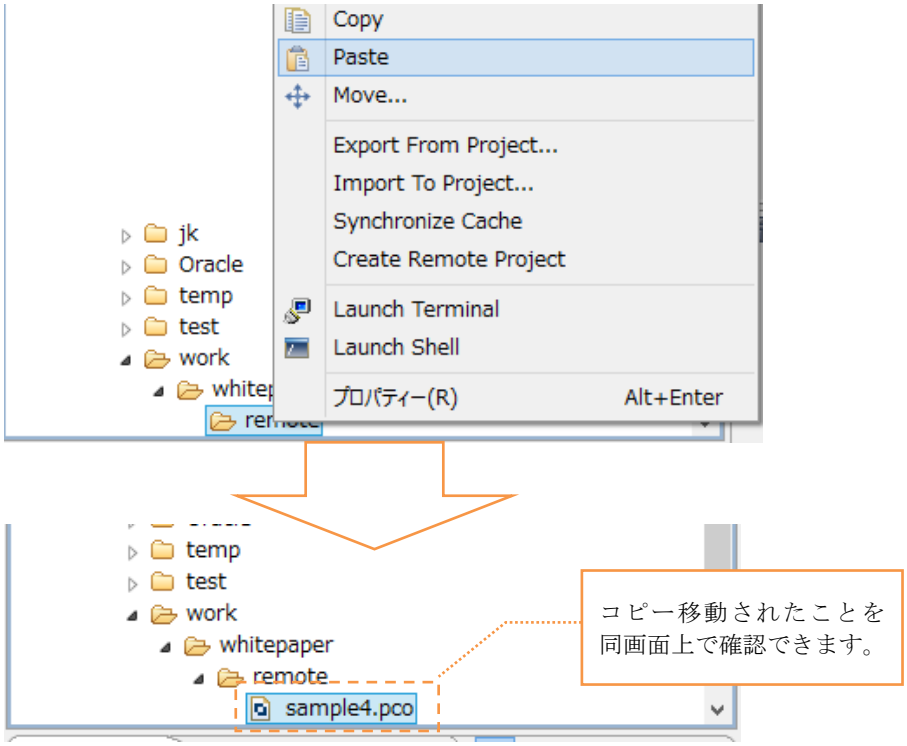
① パースペクティブを［Remote System Explorer］に切り替え



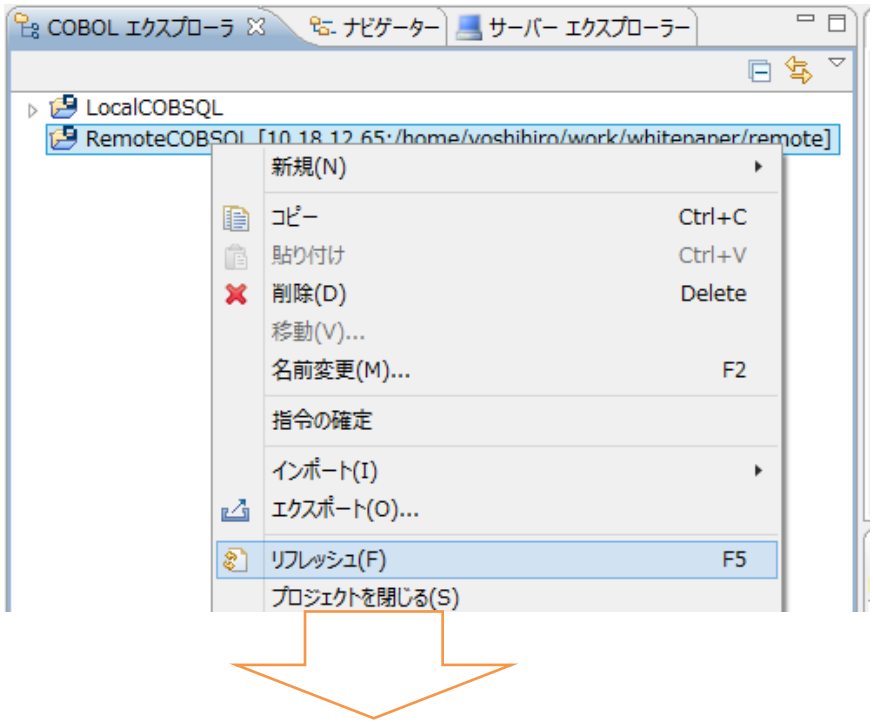

② 9) で追加した接続先（Linux）に対して「SFTP ファイル」を展開し、「sample4.pco」をコピー

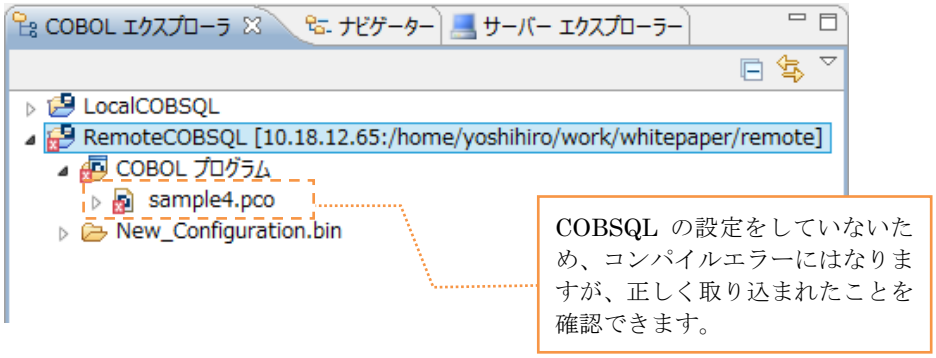


③ 同じ画面上でリモートプロジェクト用のディレクトリまでナビゲートしコピーしたファイルをペースト




④ COBOL パースペクティブに戻り、プロジェクトをリフレッシュしてプロジェクトに sample4.pco が取り込まれることを確認





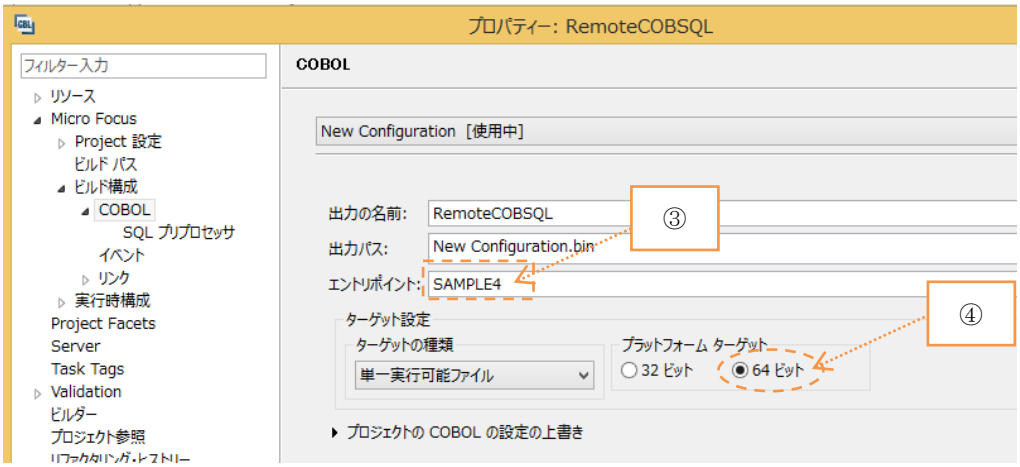

12) COBSQL の指定及び生成モジュールのエントリポイント等を指定

① プロジェクトを右クリックから［プロパティー(R)］を選択

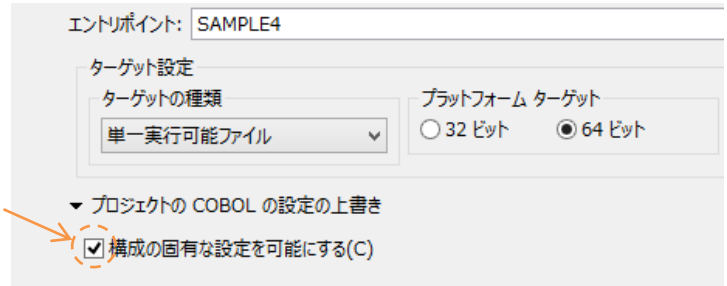
② [Micro Focus] > [ビルド構成] > [COBOL] ページに移動

③ [エントリポイント] 欄にて「SAMPLE4」を指定

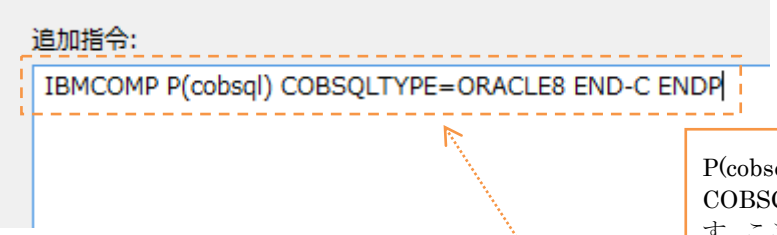
④ [プラットフォームターゲット] 欄では「64 ビット」を選択




⑤ [プロジェクトの COBOL の設定の上書き] を展開し [構成の固有な設定を可能にする] をチェック

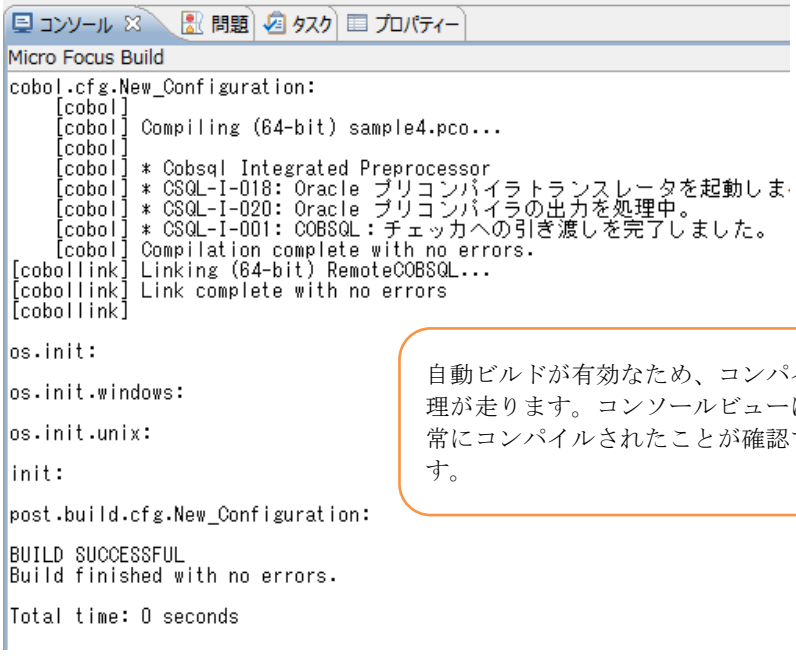


⑥ 画面を下にスクロールして［追加指令］欄に付録 1 で指定されたコンパイラ指令及び COBSQL を有効にするためのコンパイラ指令を指定 [2]

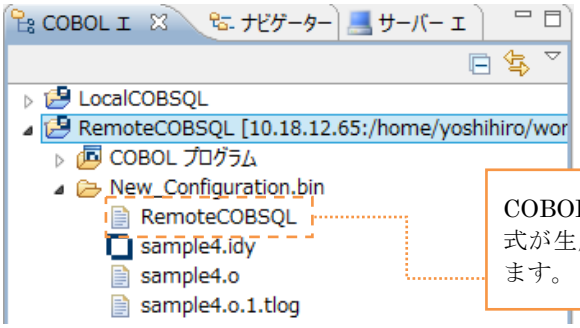



P(cobsql) 以降の指定が COBSQL 用の指令となります。ここでは ORACLE 8 以降の Pro*COBOL 向けに処理するよう指示しています。

⑦ ［OK］ボタンを押下




自動ビルドが有効なため、コンパイル処理が走ります。コンソールビューにて正常にコンパイルされたことが確認できます。

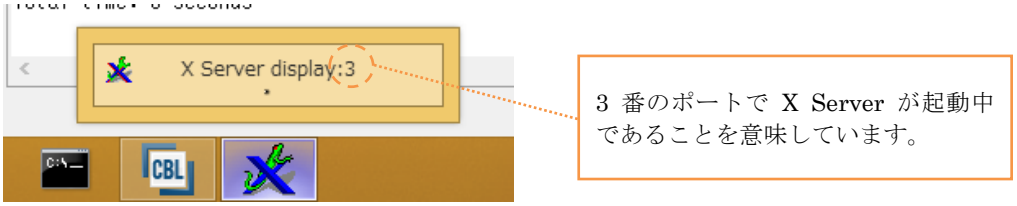



COBOL エクスプローラでも実行形式が生成されていることが確認できます。

[2] 付録 1 で使用した Oracle が提供する make ファイルでは NESTCALL を指定していました。この指令は Server Express のあるバージョンで廃止された指令であり、Visual COBOL では指定しても指定がないものと解釈しますのでここでは外しています。

13) スタートメニューから Micro Focus ViewNow X Server 9.6.4 X Server を起動

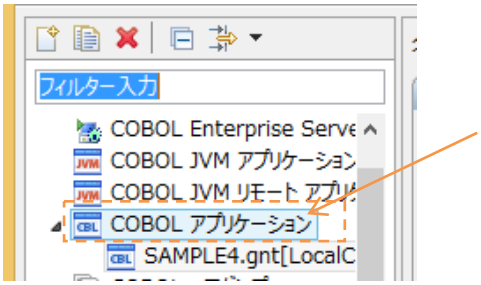
Windows のタスクバーにて起動されたことを確認できます：




14) 生成されたモジュールをデバッグ実行

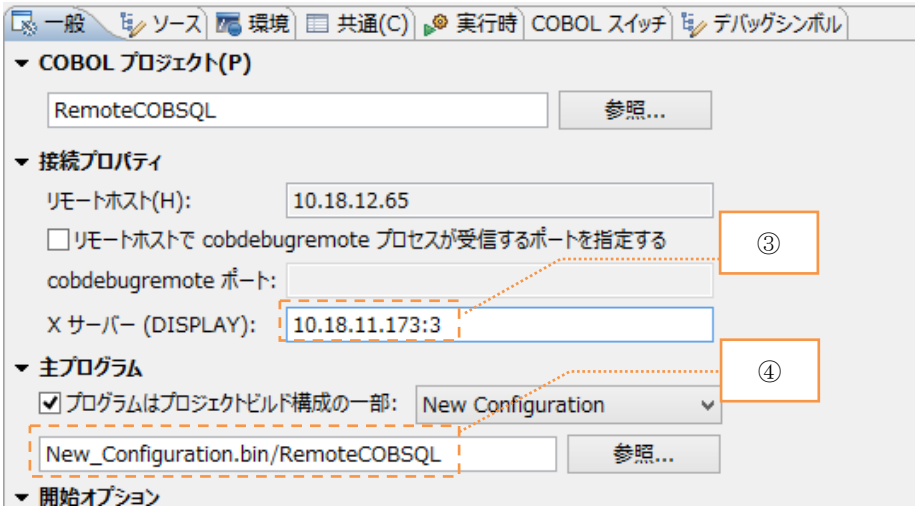
① 実行モジュールを右クリックから [デバッグ] > [デバッグの構成(B)] を選択

② [COBOL アプリケーション] をダブルクリック



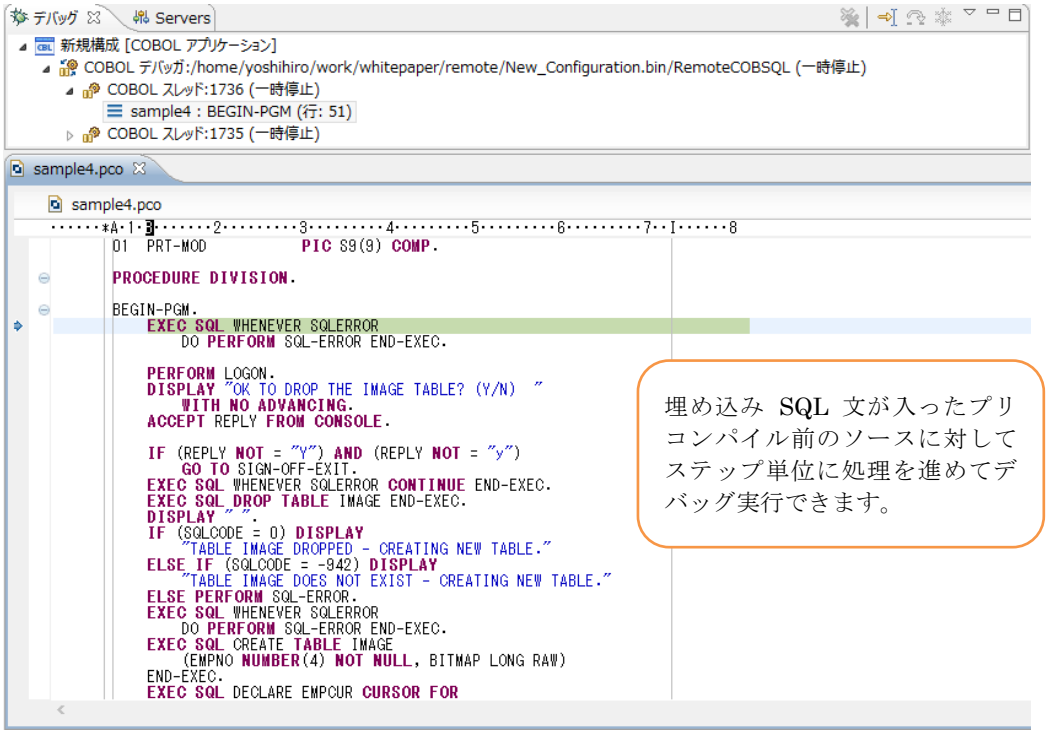

③ [X Server(DISPLAY)] 欄にて
＜Windows のホスト名/IP アドレス＞:＜13) で確認したポート＞
を入力

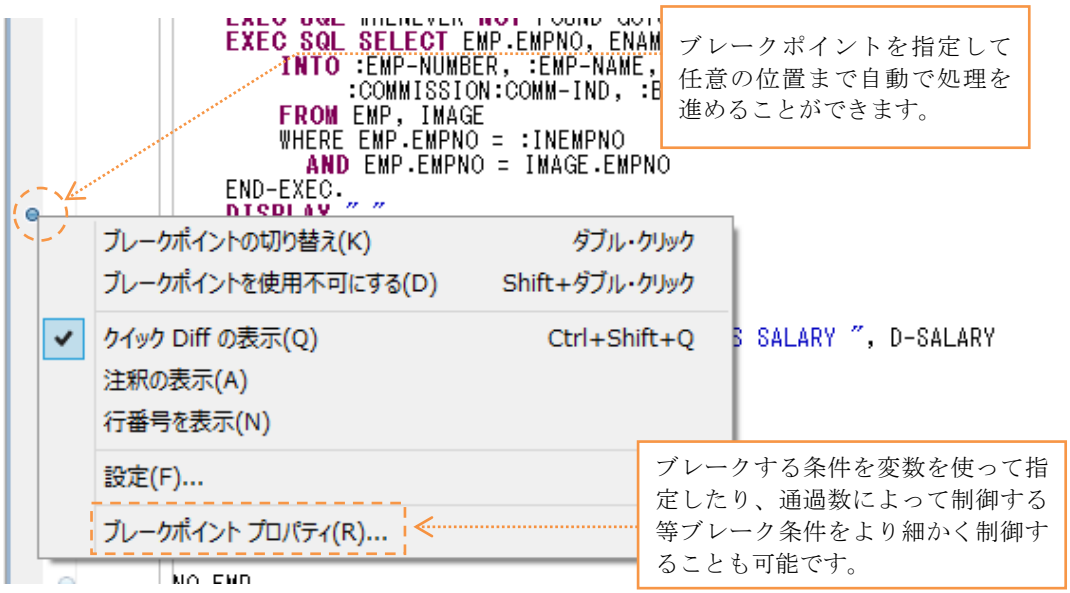
④ 主プログラム欄に生成された実行形式モジュールが指定されていることを確認し [デバッグ] ボタンを押下

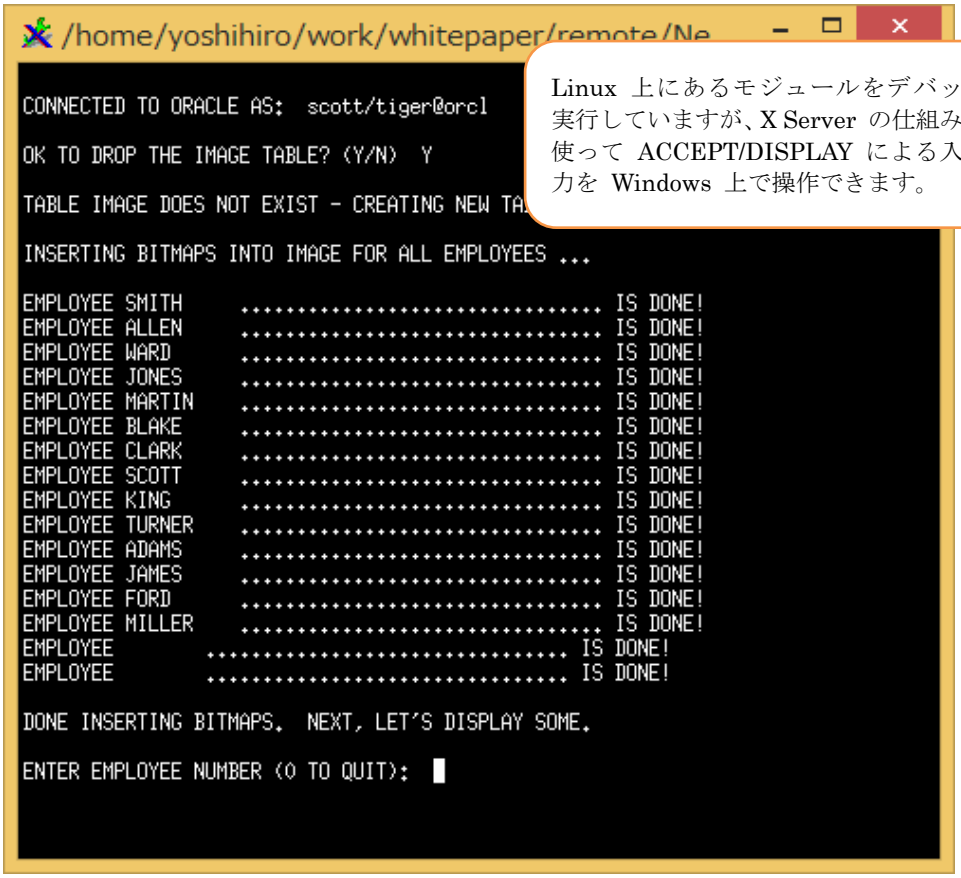


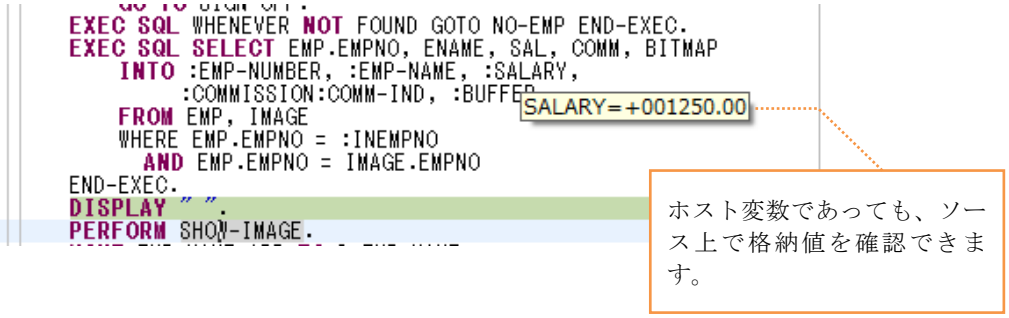
⑤　パースペクティブの切り替えに関する警告には［はい(Y)］を選択

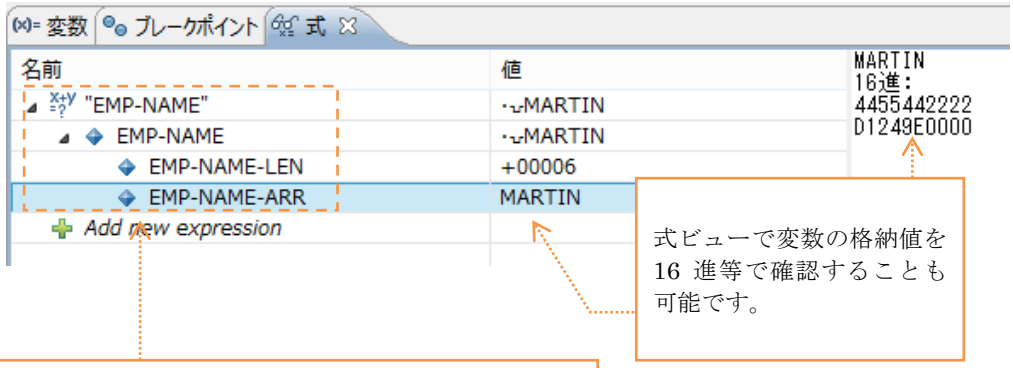
デバッグ実行が開始されます：



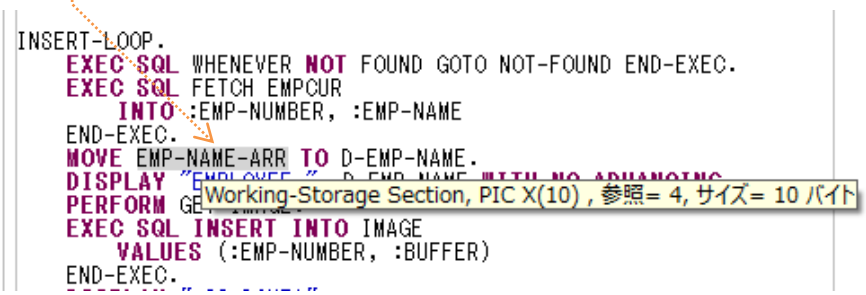

/home/yoshihiro/work/whitepaper/remote/Ne
CONNECTED TO ORACLE AS: scott/tiger@orcl
OK TO DROP THE IMAGE TABLE? (Y/N) Y
TABLE IMAGE DOES NOT EXIST - CREATING NEW TA
INSERTING BITMAPS INTO IMAGE FOR ALL EMPLOYEES ...
EMPLOYEE SMITH ..... IS DONE!
EMPLOYEE ALLEN ..... IS DONE!
EMPLOYEE WARD ..... IS DONE!
EMPLOYEE JONES ..... IS DONE!
EMPLOYEE MARTIN ..... IS DONE!
EMPLOYEE BLAKE ..... IS DONE!
EMPLOYEE CLARK ..... IS DONE!
EMPLOYEE SCOTT ..... IS DONE!
EMPLOYEE KING ..... IS DONE!
EMPLOYEE TURNER ..... IS DONE!
EMPLOYEE ADAMS ..... IS DONE!
EMPLOYEE JAMES ..... IS DONE!
EMPLOYEE FORD ..... IS DONE!
EMPLOYEE MILLER ..... IS DONE!
EMPLOYEE ..... IS DONE!
EMPLOYEE ..... IS DONE!
DONE INSERTING BITMAPS. NEXT, LET'S DISPLAY SOME.
ENTER EMPLOYEE NUMBER (0 TO QUIT):
Linux 上にあるモジュールをデバッグ実行していますが、X Server の仕組みを使って ACCEPT/DISPLAY による入出力を Windows 上で操作できます。

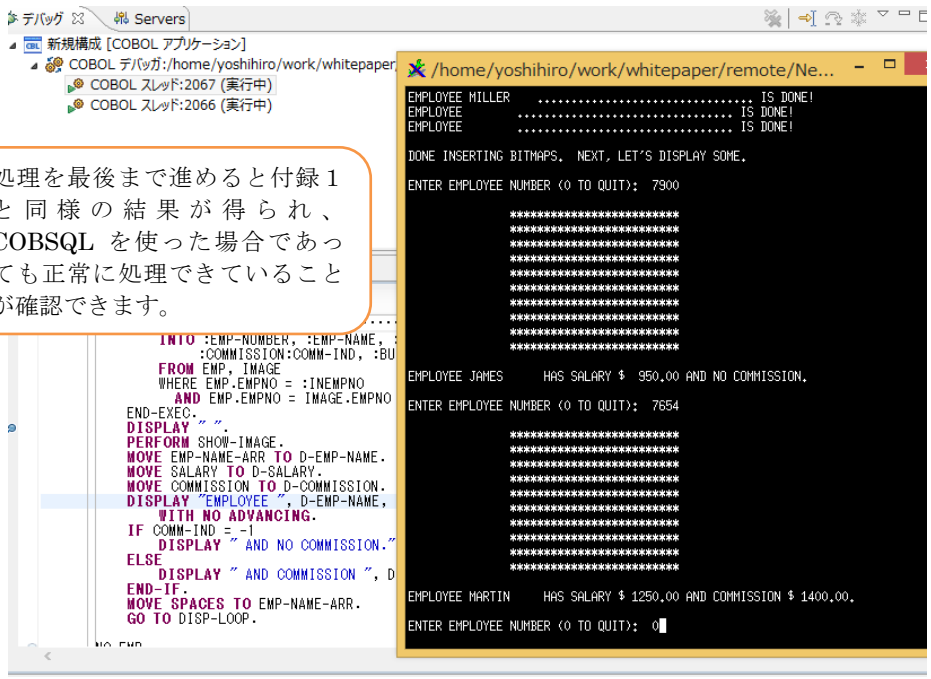

EXEC SQL WHENEVER NOT FOUND GOTO NO-EMP END-EXEC.
EXEC SQL SELECT EMP.EMPNO, ENAME, SAL, COMM, BITMAP
INTO :EMP-NUMBER, :EMP-NAME, :SALARY,
:COMMISSION:COMM-IND, :BUFFER
FROM EMP, IMAGE
WHERE EMP.EMPNO = :INEMPNO
AND EMP.EMPNO = IMAGE.EMPNO
END-EXEC.
DISPLAY " ".
PERFORM SHOW-IMAGE.
SALARY=+001250.00
ホスト変数であっても、ソース上で格納値を確認できます。




式ビューで変数の格納値を 16 進等で確認することも可能です。

EMP-NAME-LEN 及び EMP-NAME-ARR は Pro*COBOL によるプリコンパイルで EMP-NAME から展開される変数です。しかし、このプリコンパイル前のソースでも COBSQL は正式な COBOL の変数として扱うよう作りこまれています。




処理を最後まで進めると付録 1 と同様の結果が得られ、COBSQL を使った場合であっても正常に処理できていることが確認できます。

## 付録３． Linux – COBSQL(動的ロードモジュールへのビルド)

1) Linux サーバにログインして Oracle のライブラリをリンクした共有ライブラリを作成 [3]

```
$ cob -ze "" -o orainit.so `cat ~/cobopt.ora`
$
```

付録２で用意した cobopt.ora に書かれたリンク命令をこのコマンドで指定します。

2) リモート開発用のデーモンを本検証用に再設定して起動

① 付録２で起動したリモート開発用のデーモンを停止

```
# $COBDIR/remotedev/stoprdodaemon
Process 1241 located:

                                        :

Do you wish to continue and kill it? (y/n): y
Kill signal sent to process 1241
#
```

[3] 付録１でプリコンパイル展開された sample4.cob を見ると埋め込み SQL 文は下記のように Oracle が提供するライブラリ CALL 等に変換されています。実行形式の場合は Oracle のライブラリを実行形式モジュールにリンクインしてこれらへの参照を解決していました。動的ロードモジュールの場合は、ライブラリをモジュールへリンクインすることができないため、参照するライブラリをまとめた共有ライブラリを用意し、これを実行前にメモリにロードさせてこれらへの参照を解決させます。

付録１でプリコンパイルされた SAMPLE4.cbl を見ると CONNECT 文の箇所で

```
                                        :
      *    EXEC SQL SELECT EMP.EMPNO, ENAME, SAL, COMM, BITMAP
      *        INTO :EMP-NUMBER, :EMP-NAME, :SALARY,
      *             :COMMISSION:COMM-IND, :BUFFER
      *        FROM EMP, IMAGE
      *        WHERE EMP.EMPNO = :INEMPNO
      *          AND EMP.EMPNO = IMAGE.EMPNO
      *    END-EXEC.
           CALL "SQLADR" USING SQ0006 SQL-STMT
           MOVE 1 TO SQL-ITERS
           MOVE 126 TO SQL-OFFSET
           MOVE 0 TO SQL-OCCURS
           MOVE 1 TO SQL-SELERR
           MOVE 0 TO SQL-SQPMEM
                                        :
```

② COBOPT 環境変数への設定値をリセット

```
# unset COBOPT
#
```

③ 1) で用意した共有ライブラリが格納されたディレクトリを LD_LIBRARY_PATH に追加

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~yoshihiro/work/whitepaper/lib
#
```

④ リモート開発用のデーモンを起動

```
# $COBDIR/remotedev/startrdodaemon
Checking Java Version
Correct Java Version installed, proceeding
Starting RSE daemon...
Daemon running on: tok-rhel65-64, port: 4075
                                          :
```

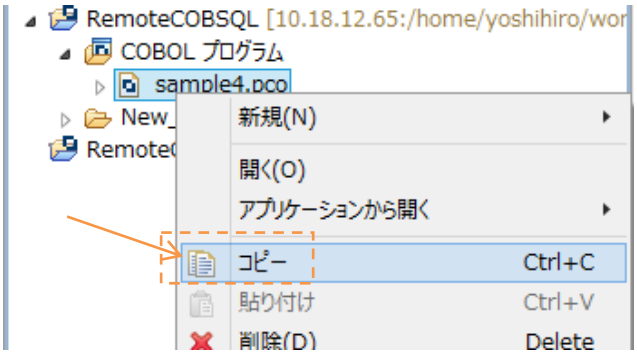
3) 付録２で使用した Eclipse ワークスペースを再起動

4) 付録２で新規作成した要領でワークスペースに COBOL リモート開発プロジェクトを追加

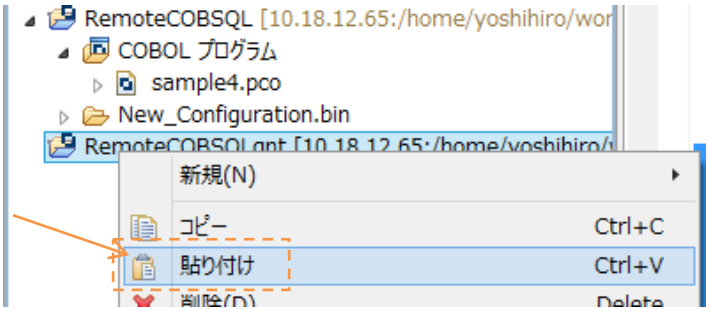
5) 追加したプロジェクトへ sample4.pco を付録２で用意したプロジェクトからコピー

① COBOL エクスプローラにて付録２で使用したプロジェクトを開く

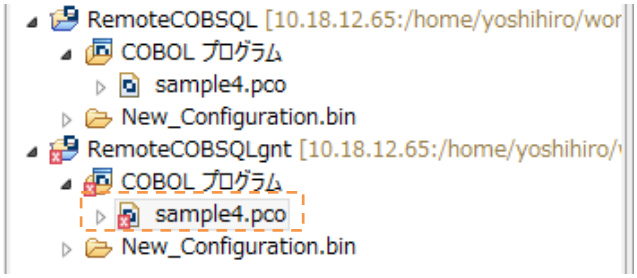
② sample4.pco をコピー



③　追加したプロジェクト上でペースト




追加したプロジェクトのディレクトリへ sample4.pco がコピーされます：



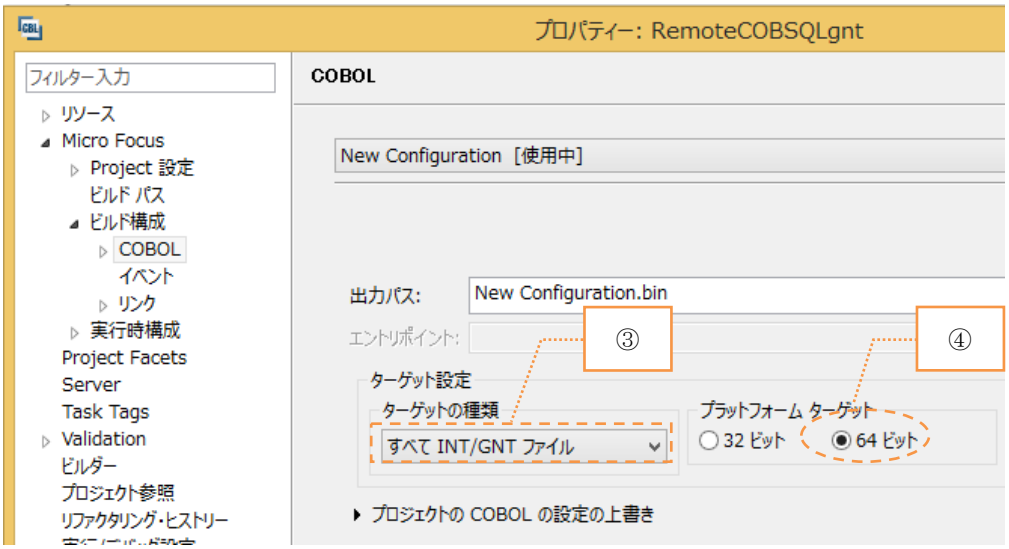

6)　COBSQL の指定及びビルドターゲット等を指定

①　プロジェクトを右クリックから［プロパティー(R)］を選択

②　[Micro Focus] > [ビルド構成] > [COBOL] ページに移動

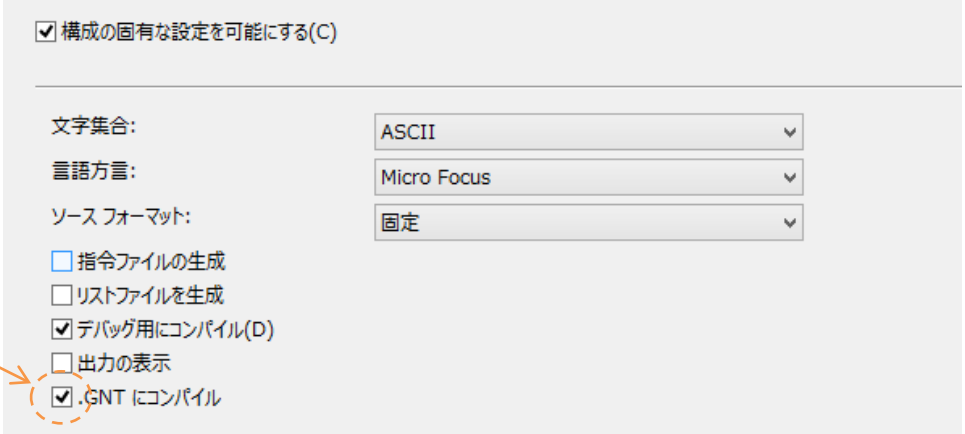
③　[ターゲットの種類] 欄にて「すべて INT/GNT ファイル」を選択

④　[プラットフォームターゲット] 欄では「64 ビット」を選択

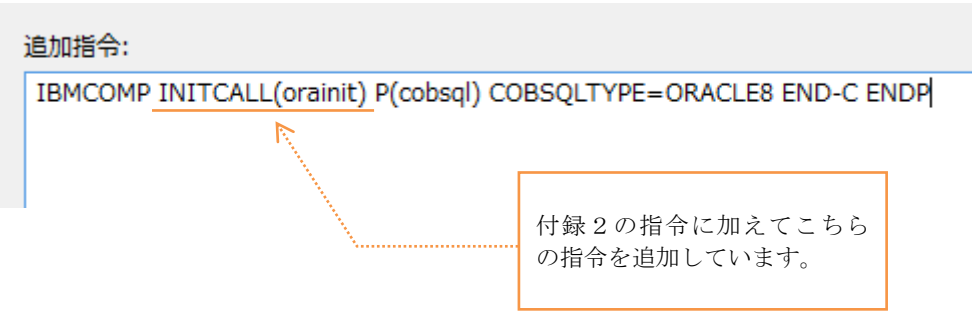


⑤ [プロジェクトの COBOL の設定の上書き] を展開し [構成の固有な設定を可能にする] をチェック

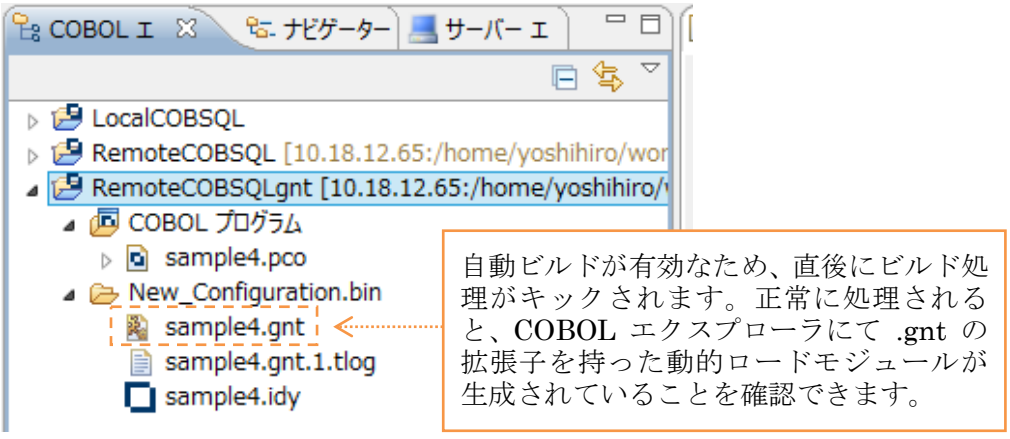
⑥ [.GNT にコンパイル] をチェック



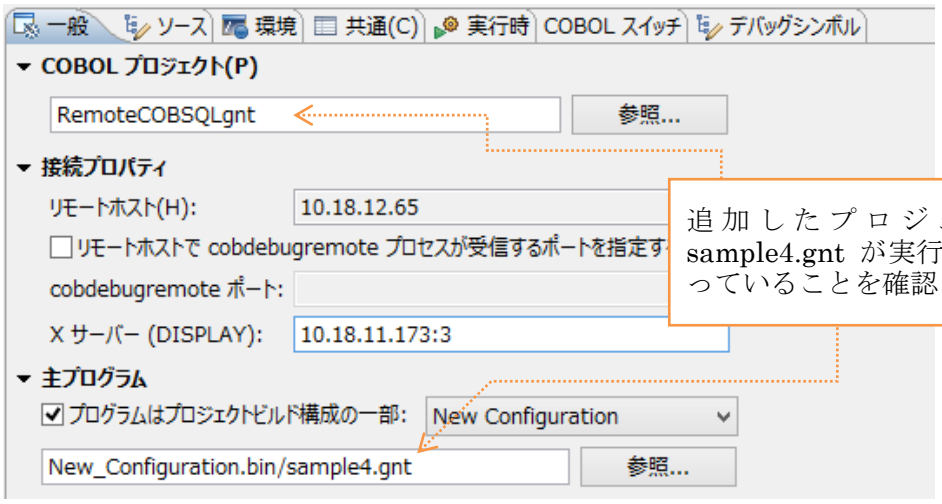

⑦ 画面を下にスクロールして [追加指令] 欄に付録 1 で指定されたコンパイラ指令、1) で用意した共有ライブラリをモジュール実行前にロードするための指令、COBSQL を有効にするための指令を指定

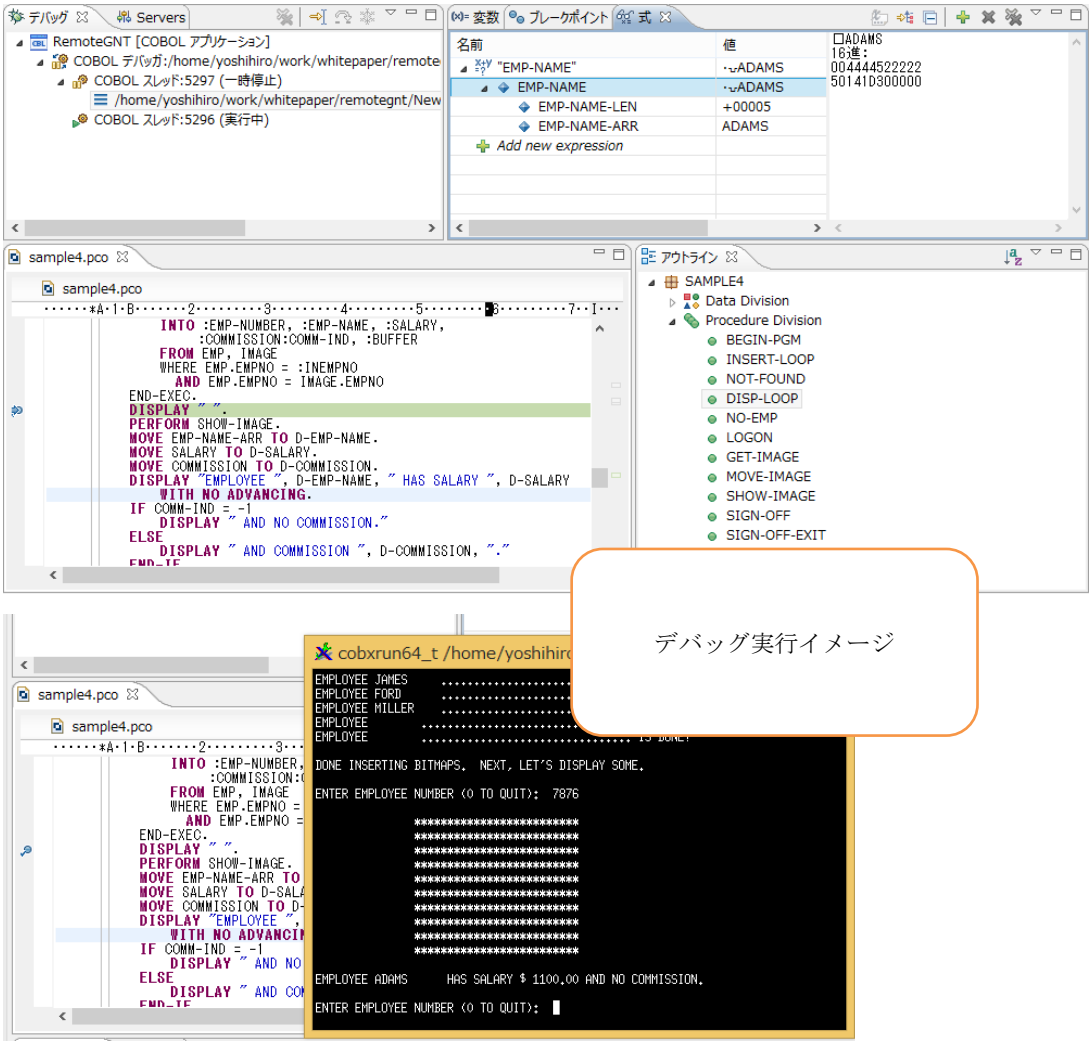



⑧ [OK] ボタンを押下



7) 生成されたモジュールを付録２の要領でデバッグ実行できることを確認




以上