Micro Focus Visual COBOL 2.3J 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

2016年04月22日

マイクロフォーカス株式会社

第2版

Copyright © 2016 Micro Focus. All Rights Reserved. 記載の会社名、製品名は、各社の商標または登録商標です。

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 (2016 年 04 月 18 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。本検証は 2015 年 12 月と 2016 年 4 月で計2回検証を行っています。1 回目は Visual COBOL 2.3J に対して、2 回目には Visual COBOL 2.3J Update 1 に対して検証しています。

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上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 Linux/UNIX 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 Linux/UNIX 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 Linux/UNIX 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

【第1回検証】

▶ Database サーバー

ソフトウェア

- Windows Server 2012R2
- · Oracle Database 12c Release 1(12.1.0.2.0) for Microsoft Windows(x64)

ハードウェア

機種:	n/a(public クラウドを利用)
CPU :	Intel Xeon CPU E5-2670 v2 2.50GHz
Memory :	3.75 Gbyte

▶ Database クライアント / Visual COBOL 開発サーバー

ソフトウェア

- Red Hat Enterprise Linux Server release 7.1
- · Oracle Database 12c Release 1 Client(12.1.0.2.0) for Linux x86-64(64bit)
- Micro Focus Visual COBOL 2.3J Development Hub

ハードウェア

機種:	Dell LATITUDE E6530
CPU :	Intel Core i5-3360M CPU 2.80GHz
Memory :	2.00 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 10 Enterprise
- Micro Focus Visual COBOL 2.3J for Eclipse
- ・ Eclipse 4.4.2(Micro Focus Visual COBOL に同梱されたものを利用)

ハードウェア

機種:	Dell OPTIPLEX 7010
CPU :	Intel Core i7-3770 CPU 3.40GHz
Memory :	4.00 Gbyte (ゲスト OS に割り当てたサイズ)

【第2回検証】

▶ Database サーバー

ソフトウェア

- Windows Server 2012R2
- Oracle Database 12c Release 1(12.1.0.2.0) for Microsoft Windows(x64)

ハードウェア

機種:	n/a(public クラウドを利用)
CPU :	Intel Xeon CPU E5-2670 v2 $2.50 \mathrm{GHz}$
Memory :	3.75 Gbyte

▶ Database クライアント / Visual COBOL 開発サーバー

ソフトウェア

- Red Hat Enterprise Linux Server release 7.2
- · Oracle Database 12c Release 1 Client(12.1.0.2.0) for Linux x86-64(64bit)
- · Micro Focus Visual COBOL 2.3J Update 1 Development Hub

ハードウェア

機種:	Dell OPTIPLEX 7010
CPU :	Intel Core i7-3770 CPU 3.40GHz
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 10 Enterprise
- Micro Focus Visual COBOL 2.3J Update 1 for Eclipse
- ・ Eclipse 4.4.2(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 i5-3360M CPU 2.80GHz
Memory :	4.00 Gbyte (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 (12.1.0.2.0) 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(12.1.0.2.0) for Linux x86-64 のインストー ラ linuxamd64_12102_examples.zip を取得
- ② linuxamd64_12102_examples.zip を解凍
- ③ <展開したフォルダ>¥stage¥Components¥oracle.precomp.companion¥12.1.0.2.0 ¥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¹ 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 2.3J、Visual COBOL 2.3J Update 1 のいずれを用いても正常にコンパイ 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、実際にプログラマがメンテナンス をする埋め込み SQL 文が入ったままのソースを直接 IDE 上で管理できることが確認で きています。これにより、Visual COBOL が COBOL 開発用に IDE に作りこんだ開発 補助機能を駆使して、Oracle データベースと連携する COBOL アプリケーション開発作 業の効率性を著しく向上させることが可能となります。

以上

¹ 付録では第1回目の検証(Visual COBOL 2.3J を用いた検証)の結果を記しています。

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

s cat samp	le4.pco	_			
01	USERNAME	PIC X(10)	VARYING.		
01	PASSWD	PIC X(10)	VARYING.		
	MOVE "scott"	: TO LISERNAME-ARI	R		
	MOVE 5 TO USE	RNAME-I FN	A.		
	MOVE "tiger"	TO PASSWD-ARR.			
	MOVE 5 TO PAS	SWD-LEN.			
	EXEC SQL				
	CONNECT :	USERNAME IDENT	IFIED BY : PASSW)	
	END-EXEC.				
	DISPLAY ″″.				
	DISPLAY "CONN	ECTED TO ORACLI	E AS USER∶″, ∣	JSERNAME-ARR.	
	DISPLAY " "				

```
編集後
```

```
$ cat sample4.pco
:
01 CONNSTR PIC X(25) VARYING.
:
MOVE "scott/tiger@cloudorcl" TO CONNSTR-ARR.
MOVE 21 TO CONNSTR-LEN.
EXEC SQL
CONNECT : CONNSTR
END-EXEC.
DISPLAY " ".
DISPLAY " CONNECTED TO ORACLE AS USER: ", CONNSTR-ARR(1:5).
DISPLAY " ".
:
```

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/pr
         ocob2/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 12.1.0.2.0 - Production on Thu Oct 22 11:54:37 2015
₩1
         Copyright (c) 1982, 2014, Oracle and/or its affiliates. All rights reserved.
         System default option values taken from: /home/yoshihiro/app/yoshihiro/produc
         t/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/yo
\times 2
         shihiro/app/yoshihiro/product/12.1.0/client_1/lib/ldflags```cat /home/yoshi
         hiro/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: 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 AS USER: scott 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²

- 2) root 権限を持ったユーザーでログイン
- 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: localhost.localdomain, port: 4075

² 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] > [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

CBL	COBOL - Eclipse				
ファイ	ル(F) 編集(E)	ナビゲート(N)	検索 プロジェク	ኑ(P)	実行(R) ウィンドウ(W) ヘルプ(H)
	新規(N)		Alt+シフト+N	> 😫	COBOL JVM プロジェクト
	ファイルを開く(.)			2	COBOL プロジェクト
	閉じる(C)		Ctrl+W	1	COBOL コピーファイル プロジェクト
	すべて閉じる(L)		Ctrl+シフト+W	षि	COBOL リモート プロジェクト
	(日本(の)		Chill C	ġ	リモート COBOL コピーファイル プロジェクト

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)
COBOL コンパイルタイプ	•••	ネイティブコード

🔤 リモート COBOL プロジェクトの新規作成	—		×
リモート coBol プロジェクト ワークスペースまたは外部にリモート COBOL プロジェクトを作成		ĺ)
プロジェクト名: RemoteCOBSQL - ファイル システム ファイル システムを選択: リモート ファイル システム (RSE) リモート ファイル システムの場合、RSE サポートによりリモート プロジェクトで作業でき ムのロケーションの指定は不要で、リモートマシン上のロケーションの指定だけが必要 ネットワーク ファイル システムの場合は、ローカルマシン上のプロジェクトの場所(マッ) クトパ ファビートマシン トのプロジェクトの場所(マッ)	ます。ローカル です。 プされたドライブ	ファイルシ: 、上のプロ:	<
 コンパイルタイプ COBOL コンパイルタイプ: ネイティブ コード コンパイルタイプで "ネイティブ コード" を選択すると、リモートマシンのプロセッサ上に ドにコンパイルされます。 コンパイルタイプで "JVM バイトコード" を選択すると、すべての JVM 互換の環境で シンコードにコンパイルされます。 	に直接実行可 で実行可能な	能なマシン Java 仮想	✓ <- !7
⑦ < 戻る(B) 次へ(N) > 終日	7 (<u>F</u>)	キャンセ	JL

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

🔤 リモート COBOL プロジェクトの新規作成			×	
リモート coBol プロジェクト ワークスペースまたは外部にリモート COBOL プロジェクトを作成				
プロジェクト名: RemoteCOBSQL リモート設定			,K	
接続名: リモート [✓ 接続	の新規作 ▲ Brow	成 se	
リモート ロケーションはリモート マシンのプロジェクト パスに設定しなければいけません。				

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

New Connection	_		×
Select Remote System Type		П	
Micro Focus DevHub - SSH プロトコルによるリモートファイルシステム(RSE)の ス	のファイルアク	⁷ 2 =	
System type:			
プルタ入力			
✓ General ☐ Micro Focus DevHub (RSE 経由) ☐ Micro Focus DevHub SSH 使用			

③ [Host name] 欄に Linux 環境のホスト名もしくは IP アドレスを入力して [終了] ボタンを押下]

New Connection	_	×
Remote Micro Focus Define connection inf	s DevHub (RSE 経由) System Connection ormation	
Parent profile :	DESKTOP-4BQMBC1	~
Host name: Connection name: Description:	10.18.11.118 <	 ~

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

			~	
リモート coBol プロジェクト ワークスペースまたは外部にリモート COBOL プロジェクトを作成		ĺ		
プロジェクト名: RemoteCOBSQL リモート設定				
接続名: 10.18.11.118 リモート [リモート 「 リモート 「ケーションはリモート マシンのプロジェクト パスに設定しなければいけません。	✓ 接	続の新規化 - A Brow	F成	

② 「My Home」を展開

Browse For Fold	er	×
Select a folder		
My Home	クリック	
(>) ♣ My Home		
`≶ ‡⇒ Root		

③ Linux 側のユーザーログイン情報を指定して [OK] ボタンを押下

Enter Password		×	
System type: Host name: Connection name: <u>U</u> ser ID: <u>P</u> assword (optional):	Micro Focus DevHub 10.18.11.118 10.18.11.118 yoshihiro	。(RSE 経由)	
		本チェック にチェック おきます。	ウボックス クを入れて
[<u>О</u> К ‡	ヤンセル(<u>A</u>)	

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

Browse For Folder	×
Select a folder	
My Home	
✓ [⇒] My Home	~
> 🧰 app	
> 🧰 oralnventory	
> 🧰 temp	
> 🧰 test	
> 🗀 training	
> 🗀 tutorial	
🗸 🗁 work	
🗸 🗁 wp	
> 🧰 rmtprj	

⑤ [終了] 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 ログラムヘコピー
 - COBOL エクスプローラにてプロジェクトを右クリックし、 [インポート] > [インポート]

を選択

と思わ	N.				
COBC)L - E	clipse			
ファイル(F)	編創	集(E) ナビゲート(N) 検索 ブ	ロジェクト(P) 実行(R) ウィント	^ะ ウ(W	N) ヘルプ(H)
📑 🗝 🖪	R	👜 💠 • 🜔 • 💁 • 🍅	0 🖨 🛷 🕶		℃
		新規作成(N)			
		שצ-	Ctrl+C		
🕗 R	Ē	貼り付け	Ctrl+V		
	×	削除(D)	削除		
	<u>.</u>	Remove from Context	Ctrl+Alt+シフト+下へ		
		移動(V)			
		名前を変更(M)	F2		
		ビルド アクション		>	
		指令の確定			
		コード分析		,	
		インポート(1)		÷ 🕆	を、 リモート プロジェクト
	n a	エクスポート(O)			☆ ローカル Micro Focus プロジェクトのリモート プロジェクトへの変換
	-			-	
	<u>«</u>	史新(ド)	F5	1	Net Express ノロシェクトの変換
		ノロンエクトを閉しる(5) 毎日ダカゴロジェクトを用じる(4	D	2	≦ インポート(I)
<u> </u>			<i>n</i>	1 C	

② [Remote file system] を選択の上、[次へ] ボタンを押下

「風」 インポート	_		\times
選択 Import resources from a remote file system		Ľ	5
インポート・ソースの選択(S): フィルタ入力			
 > Dava EE > Dava EE > Maven > Micro Focus > Micro Focus インターフェイスマッパー > Common Remote Systems ↓ Bemote Systems ↓ Remote file system ↓ Tasks 			^

③ [Browse] ボタンを押下

import	- 🗆 X
Remote file system Import resources from a remote file system.	
From directory :	V Browse

④ [Connection] 欄にてプロジェクトを作成した際に作成した Linux サーバー への接続を選択

Browse Fo	r Folder	×
Select a folde	r	
Connection:	Local 🗸	New
My Home	Local 10.18.11.118	
> ∔ My H > ∔ Drive	Home es	

⑤ sample4.pco が格納された付録1で作業したディレクトリまでナビゲートし、[OK] ボタンを押下



⑥ 一度 [Deselect All] ボタンを押下し、全ての選択を解除した後に、
 sample4.pco のみにチェックを入れ、[終了] ボタンを押下

import							×
Remote file system	n om a remote file	system.					
From directory:	DESKTOP-4BQN	1BC1.10.18.11.118	:/home/yoshih	iro/app/yoshi e4.o e4.pco e5.pco e6.pco e7.pco e8.pco e8.pco e9.pco ache.pco	hiro/pr ∨	Brows	e
Select T	ypes	Selec	t All		Deselect A		~ _
 COBOL - Ecli ファイル(F) 編集(ご マ □ □ □ □ 	ipse (E) ナビゲート ● 参 ▼ ○	(N) 検索 フ ▼ Q ▼ (2)	[終了] したフ 取り込	ボタンを ァイルが まれます。	押下する プロジェ	っと選打 クトル	尺こ
 ♀ ♀ Period ♀ 	宅 ・ ナビゲ … COBSQL [10.18 DL プログラム ample4.pco _Configuratio	■ サーバ	□ =/yoshihi				

12) COBSQL を有効化

- ① プロジェクトを右クリックから [プロパティ] を選択
- ② [Micro Focus] > [ビルド構成] > [COBOL] > [SQL プリプロセッサ] ページに移動

③ 【構成の固有な設定を可能にする】及び [SQL プリプロセッサの使用] にチェック 入れ、[プリプロセッサの種類] 欄にて「Oracle Pro*COBOL(COBSQL)」を選択

🔤 プロパティ: RemoteCOBSQL			— 🗆 X	
ንብሥንአታ	sqL プリプロセッサ		⟨→ -	
 > リソース Micro Focus ビルドパス > ビルド構成 > COBOL SQL ブリプロセッサ コード分析 	New Configuration [使用中]		◆ 構成の管理 ^	
xiandD797日を99 イベント > リンク > プロジェクト設定 > 実行時構成 Project Facets Server > Task Repository	CON C プリプロセッサの使用 プリプロセッサの種類: OpenESQL 描令: Oracle Pro*CC フィルタテキストを入OpenESQL come	BOLCOBSOL I	~	

④ COBSQL 指令「COBSQLTYPE」に「ORACLE8」を指定³

リプロセッサの種類: Oracle Pro*COBC	DL(COBSQL)	
指令:		
フィルタテキストを入力		
設定	値	
COBSQLTYPE	ORACLE8	
CSTART		
CSTOP	ORACLE	
	ORACLE8	
DEBUGFILE		

13) コンパイラ指令及び生成モジュールのエントリポイント等を指定

① [Micro Focus] > [ビルド構成] > [COBOL] ページに移動

³ Oracle 8 以降の Oracle と連携する場合の指定となります。

- ② [エントリポイント] 欄にて「sample4」を指定
- ③ 【プラットフォームターゲット】欄では「64 ビット」を選択

ንብሥንአታ	COBOL
 > リソース ✓ Micro Focus ビルド パス ✓ ビルド構成 	New Configuration [使用中]
 > COBOL イベント > リンク > プロジェクト設定 > 実行時構成 	出力の名前: RemoteCOBSQL 出力パス: New Configuration.bin エントリポイント: sample4 3
 Project facets Server Task Repository Task Tags Validation 	ターゲット設定 ターゲットの種類 単一実行可能ファイル

④ [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

ターゲット設定 ターゲットの種類 単一実行可能ファイル 〜	プラットフォームターゲット ○ 32 ビット
・プロジェクトの COBOL の設定の上書き	

⑤ 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を 指定4

回復可能なエラーを含める(レベル E)	~
100	
	回復可能なエラーを含める(レベル E) 100

⁴ 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。

⑥ [OK] ボタンを押下

V Dew_Configuration.bin

🚺 sample4.idy

sample4.o

📄 RemoteCOBSQL 🔚



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。 14) 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:



ViewNow X が使用するポートはタスクマネージャー等で確認ができます:



15) 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] を選択
- ② [COBOL アプリケーション] をダブルクリック

ጋ ィルタ入力	
Android Application	^
Jug Android JUnit Test	
🗄 Apache Tomcat	
🍇 COBOL Enterprise Server	
搣 COBOL JVM アプリケーション	
쨰 COBOL JVM リモート アプリケーション	
🗟 COBOL アプリケーション	
- 🛐 COBOL コアダンプ	

③ [X Server(DISPLAY)] 欄にて

<Windows のホスト名/IP アドレス>:<14) で確認したポート> を入力

💀 COBOL コアダンプ 🛜 COBOL のアプリケーションのアタッチの待村	Cobaebugremote ハ៶= Ⴡ: X サーバー (DISPLAY):	3000 10.18.11.96:	3
C:¥Windows¥System32¥cmd.exe			
/. # ㅋ ㅣ ㄱ / ㅋ / . # ㅋ			14)で確認したポ ート
キーウネット アメフメー オーウネッ 接続固有の DNS サフィックス リンクローカル IPv6 アドレス IPv4 アドレス サブネット マスク デフォルト ゲートウェイ	· · · · · · · · · · · · · · · · · · ·	96	

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

☑ プログラムはプロジェクトビルド構成の一部:	New Configuration	~	
New_Configuration.bin/RemoteCOBSQL		参照	
開始オプション			
コマンド行引数:			
		\sim	
		×	
作業ティレクトリ:			
		参照	
デバッグ オプション			
	適用(Y)	前回保管した	状態に戻す(V
			

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:

🗱 デバッグ 🛛 🦓 Servers 💃 🛛 🕾 🌾 🛛 🖘 🗖 🗖	(x)= 変数 SS ● ブレークポイント
✔ 🔤 新規構成 [COBOL アプリケーション]	名前值
✓ 爺 COBOL デパッガ:/home/yoshihiro/work/wp/rmtprj/New_C	CONNSTR-ARR (1:5) scott
✓ ● COBOL スレッド:8408 (一時停止)	
sample4 : LOGON (17: 130)	scott
→ COBOL スレッド:8407 (一時停止)	10度; 76677
	33F44
< >	<
sample4.pco 🖾	
R annulation	
sample4.pco	埋め込み SQL 文が入ったプリ
DISPLAY "NOT A VALID EMPLOYEE NUMBER	- TRY AGA コンパイル前のソースに対して
GO TO DISP-LOOP.	ステップ単位に処理を進めてデ
⊖ LOGON.	「小がす」「「「「「「」」」
MOVE SCOLL/LIGENWCIOUDOPCI TO CONN MOVE 21 TO CONNSTR-LEN.	SIR-ARK. NUV XII Caly.
EXEC SQL	
CONNECT : CONNSTR	
DISPLAY " ".	
DISPLAY "CONNECTED TO ORACLE AS USER	: ", CONNSTR-ARR(1:5).
DISPLAY " ".	
PERFORM MOVE-TMAGE	
VARYING INDX FROM 1 BY 1 UNTIL I	NDX > 8192.
MOVE-IMAGE.	
STRING ** DELIMITED BY SIZE	¥
<	>
EXEC SOL SELECT EMP.E	IPNO, ENAME, SAL, COMM, BITMAP
INTO :EMP-NUMBER.	: EMP - NAME
:COMMISSION:	OMM-TND. ブレークポイントを指定して
FROM EMP. TMAGE	任意の位置まで自動で処理を
WHERE EMP. EMPNO =	:TNEMPNO 進めることができます。
AND EMP. EMPNO =	IMAGE, EMPI









付録3. Linux – COBSQL(動的ロードモジュールへのビルド)

1) Linux サーバーにログインして Oracle のライブラリをリンクした共有ライブラリを 作成⁵

\$ cob -ze "" -o orainit.so \$	`cat ./cobopt.ora`	
	付銅 たリ す。	2で用意した cobopt.ora に書かれ ンク命令をこのコマンドで指定しま

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 5737 located:
:
Do you wish to continue and kill it? (y/n) : finished on port 48829
У
Kill signal sent to process 5737
; #

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 O TO SQL-OCCURS
	MOVE 1 TO SQL-SELERR
	MOVE O TO SQL-SQPMEM
	· · · · · · · · · · · · · · · · · · ·

⁵ 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。

COBOPT 環境変数への設定値をリセット

# unsei	t CORONI			
#				

 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:`pwd`
#

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: localhost.localdomain, 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

* • • •		新規作成(N)	al conclusion			>
	D_	כאל-			Ctrl+C	
🔁 СОВ 💈	Ē	貼り付け			Ctrl+V	
	×	削除(D)			削除	
🗸 🕗 Remo	Ð	Remove from Context		Ctrl+Alt+シフト	·+下^	
🗸 🔁 o		移動(V)				
> 🖻		名前を変更(M)			F2	
🖉 Remo		ビルド アクション				>

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 を有効化
 - ① プロジェクトを右クリックから [プロパティ]を選択
 - ② [Micro Focus] > [ビルド構成] > [COBOL] > [SQL プリプロセッサ] ページに移動
 - ③ [構成の固有な設定を可能にする] 及び [SQL プリプロセッサの使用] にチェック 入れ、[プリプロセッサの種類] 欄にて「Oracle Pro*COBOL(COBSQL)」を選択
 - ④ COBSQL 指令「COBSQLTYPE」に「ORACLE8」を指定

プリプロセッサの種類:	Oracle Pro*COBOL(COBSC	QL)	
指令:			
フィルタテキストをス	h		
2007 101 27			
設定		値	
		ORACLE8	~
COBSQLIYPE		OTTACEED	

- 7) コンパイラ指令及びビルドターゲット等を指定
 - ① [Micro Focus] > [ビルド構成] > [COBOL] ページに移動
 - ② [ターゲットの種類] 欄にて「すべて INT/GNT ファイル」を選択
 - ③ [プラットフォームターゲット] 欄では「64 ビット」を選択

🔤 プロパティ: RemoteCOBSQLGnt	
<u> </u> 7 ብ ሥራ አ	COBOL
 > リソース Micro Focus ビルド パス ビルド構成 COBOL SQL プリプロセッサ コード分析 追加のプリプロセッ 	New Configuration [使用中] 出力パス: New Configuration.bin
イベント > リンク > プロジェクト設定 > 実行時構成 Project Facets Server	エントリポイント: ② ③ ターゲット設定 ・ ・ ターゲットの種類 ・ ブラットフォームターゲット ・ すべて INT/GNT ファイル 〇 32 ビット (● 64 ビット)

- ④ [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
- ⑤ [.GNT にコンパイル] をチェック

文字集合:	ASCII	~
言語方言:	Micro Focus	~
ソース フォーマット:	固定	~
□指令ファイルの生成	□ - ド;	カバレッジを有効にする
□リストファイルを生成	רסל 🗌	イラを有効にする
デバッグ用にコンパイル(D)	田力の)表示

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、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を指定



8) 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

🗔 一般 🛛 与フース 🚾 環境 🔲 共通(C) 🔎 実行時 COBOL スイッチ 🔭								
▼ COBOL プロジェクト(P)								
R	RemoteCOBSQLGnt			参照				
▼ 接続プロパティ								
J.	モートホスト(H):	10.18.11.118	<u>) (</u>					
	□ リモートホストで cobdebugremote プロセスが受信 sample4 gnt が実行される構成に							
co	obdebugremote ポート:	8000	って	いることを確認				
Х	サーバー (DISPLAY):	10.18.11.96:3						
▼ 主プログラム								
✓ プログラムはプロジェクトビルド構成の一部: New Configuration ~								
New_Configuration.bin/sample4.gnt 参照								



以上