White Paper
Micro Focus Enterprise Developer

Micro Focus Enterprise Developer を使用したクロス開発

IBM メインフレームは現在でも日々世界のビジネストランザクションの主要な部分を担って稼働し続けています。しかしながら、一般にメインフレームアプリケーションの保守には多くの人的・コンピューティング資源を必要とし、アプリケーションの品質を確保するために多大なランニングコストを必要とするのが現実です。

Micro Focus Enterprise Developer に内蔵されている COBOL コンパイラーは、IBM メインフレームのクロス開発で約半世紀にわたって活用されてきた実績があり、各開発者が Enterprise Developer を利用することで COBOL アプリケーション保守によるランニングコストの削減が望めます。
また、テスト時に使用する開発用実行環境 Enterprise Server が内蔵する COBOL ランタイムは、メインフレーム上の実稼働環境で使用する IBM 製 COBOL コンパイラーの各種バージョンに対して個別に高い互換性を持ち、JCL、CICS、IMS についてもメインフレームと互換性のあるテスト環境を提供しています。

この文書では、Enterprise Developer を使用したクロス開発について解説します。

White Paper
Micro Focus Enterprise Developer

目次

1. 問題点の把握

メインフレーム開発業務であるアプリケーションのメンテナンスやコンパイル、テストなどにおいて、次のような問題点はないでしょうか。

問題点１：メインフレーム資源の不足

品質担保のためにテスト用区画の増設が必要になるなど資源が不足する。

問題点２：開発作業による CPU 使用率の上昇

開発作業の集中により CPU 使用率が上昇してしまいコストが高くなる。

問題点３：開発作業によるトラフィックが集中

開発作業の集中により工程に遅延が発生する。

問題点４：次世代への継承

メインフレームの経験者が少なく、企業の知識が蓄積されたアプリケーションの継承が困難である。

これらの問題を解決しようと、メインフレームからオープン環境へアプリケーションの移行を計画したが、オープン環境の OS は何か良いのか、どのように誰が管理するのか、運用形態はどのように変化するのか、現在と同じように COBOL を使用できるのかなど、移行計画時に様々な疑念が表面化し、調査はしたものの未だ何も進展がない状況へ陥ってはいないでしょうか。

2. 解決策の選択肢

現状の問題点に対して解決策を求められた場合に挙げられる選択肢は以下の方法が一般的です。

解決策１：新たな区画や資源の確保

コストはかかるが現状を維持しながら不足している資源を補充していく方法。時間をかけずに資源不足は解消できるが、前述の他問題点の解決は望めない。

解決策２：リホスト

現在のアプリケーションを出来るだけ修正せずにオープン環境へ移行して開発や運用のコスト削減を目指す方法。移行計画や設計、新旧比較テストなど一定の時間が必要になるが、段階的にモダナイゼーションも視野に入れることができ、前述の全問題点の解決が望める。

解決策３：リライト

現在のアプリケーションを全面的に刷新して別言語に書き換えるなどの方法。現行の仕様把握から新機能の設計、全テストなど多大な時間が必要になる。企業のワークフローを変更してツールを導入することも含み大規模な刷新となるが、前述の全問題点の解決が望める。

解決策４：クロス開発

開発とテストはオープン環境で実施し、メインフレームの資源使用を出来るだけ抑えることによりコスト削減を目指す方法。クロス開発方法が浸透するまでの時間が必要になるが、リホストやリライトと比較すれば最小限に抑えられる。オープン環境で開発するため、次世代への継承は見込める程度になるが、前述の他問題点の解決は望める。

本書はリホストでは敷居が高く進展が望めない状態を打開するため、クロス開発を実施してメインフレーム資源の使用を抑えたコスト削減を目指す方法を解説するものです。

3. Enterprise Developer ができること

Micro Focus Enterprise Developer は JCL, CICS, IMS エミュレーション機能を持ち、リホストも可能な製品です。また、この製品は Micro Focus Visual COBOL も包括しているため、将来的には Java や .NET などの最新技術を使用しつつも COBOL ロジックを維持することができます。

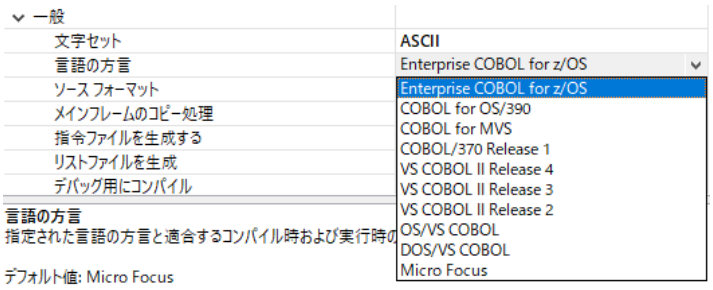
各機能の詳細に関しては、製品マニュアルページからご利用になるバージョンを選択後、内容をご確認ください。

https://www.microfocus.co.jp/

[サポート] > [COBOL・エンタープライズ製品のカスタマーケア：詳細をみる] > [製品マニュアル]

- IBM メインフレーム方言が指定可能

  コンパイル時に IBM メインフレーム OS によるコンパイラー指令方言[1]を指定できます。これにより方言に沿った指令が暗黙的に指定され、メインフレームとの高い互換性を保つことが可能となります。



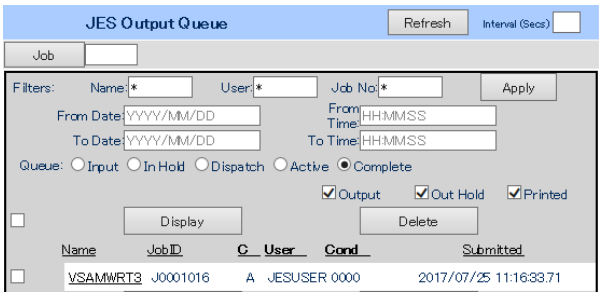
◎　クロス開発における利点

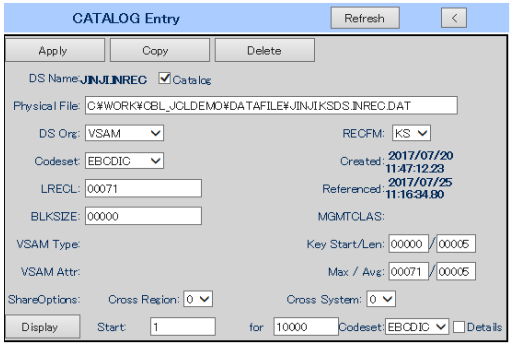
オープン環境でコンパイルエラーを検出し、
ソース修正できるため、メインフレーム資源の使用削減に貢献します。

[1] 製品マニュアル）[Micro Focus Enterprise Developer] > [リファレンス] > [コンパイラ指令] > [Compiler Directives - Alphabetical List] > [DIALECT]

- JCL が利用可能

JCL エミュレート機能によりメインフレームで使用していた JCL をオープン環境でも実行できます。一般的に使用される IBM JCL ユーティリティの大部分をサポートしており、スプール、カタログファイル、プロシージャ、世代管理ファイルもサポートしています。





◎ クロス開発における利点

オープン環境でテストが実施できるため、メインフレーム資源の使用削減に貢献します。

- CICS 構文が利用可能

CICS エミュレート機能により、EXEC CICS 構文や、一般的に使用される API, SPI の大部分をサポートしています。3270 端末を開発用である Enterprise Server インスタンスが持つリスナーポートへ接続するだけでメインフレームと同じ画面が表示されます。BMS 定義もサポートしています。

```
EXEC CICS SEND
          MAP('ACCTMNU')
          MAPSET('ACCTSET') FREEKB
          ERASE MAPONLY
END-EXEC
EXEC CICS RETURN TRANSID('AC01') END-EXEC
```

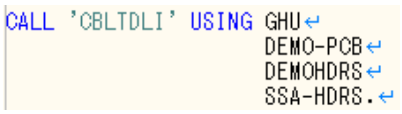
```
ACCTMNU DFHMDI SIZE=(24,80)
        DFHMDF ATTRB=(ASKIP,NORM),
               COLOR=TURQUOISE,
               LENGTH=36,
               POS=(1,20),
               PS=8,
               INITIAL='＊＊ 顧客ファイルメンテナンス ＊＊'
```

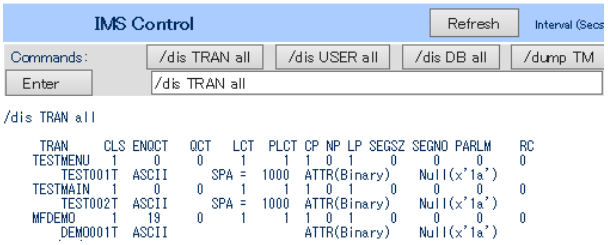
◎ クロス開発における利点

オープン環境でテストが実施できるため、メインフレーム資源の使用削減に貢献します。

- IMS DB が利用可能

  IMS エミュレート機能により、IMS プログラム構文をサポートしており、データのセグメント構成も保持できます。MPP, BMP, DLI もサポートしています。ただし、実データファイルは Micro Focus 形式のファイルへ変換する必要があります。





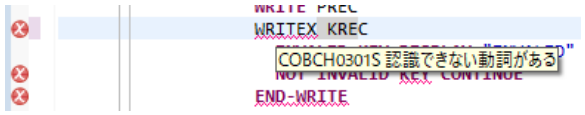
◎　クロス開発における利点

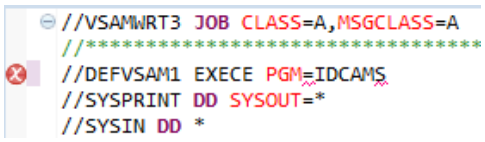
CBLTDLI 構文を使用できるためオープン環境でコンパイルエラーを確認できます。

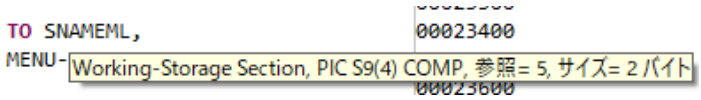
オープン環境でテストが実施できるため、メインフレーム資源の使用削減に貢献します。

- 開発支援機能の利用が可能

  COBOL エディター、JCL エディターを使用することにより、リアルタイムに入力エラーを検出できます。項目定義位置へジャンプすることや、マウスオーバーによる項目定義の確認、転送先候補の表示も可能です。また、同梱されているデータツールを使用することにより EBCDIC 文字コードデータを可視化できます。







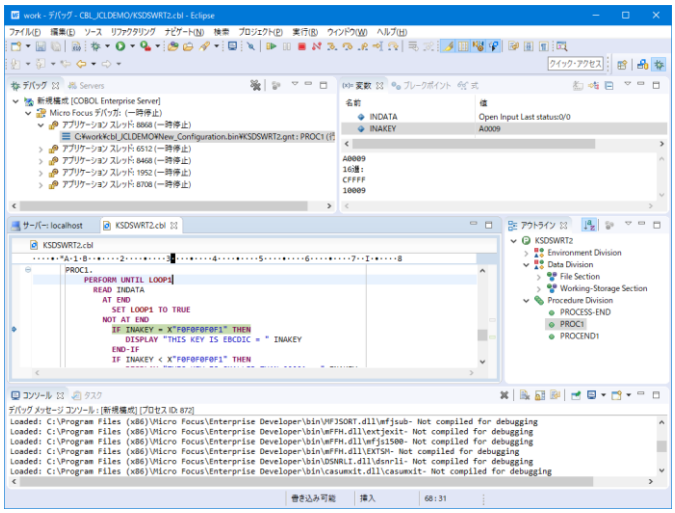
◎　クロス開発における利点

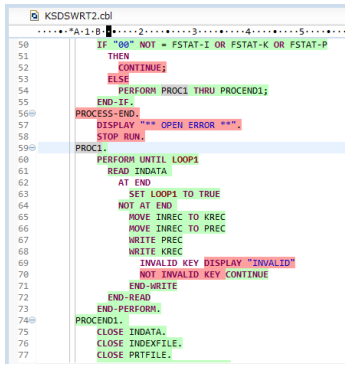
コーディングエラーやタイプミスを未然に防ぐことができるため、開発工数の削減に貢献します。

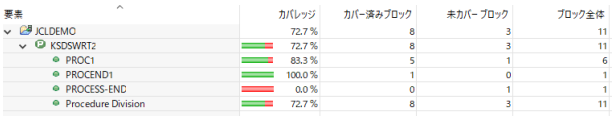
テストで出力された EBCDIC 文字コードデータを確認できるため、メインフレーム資源の使用削減に貢献します。

- ステップ実行デバッグが可能

COBOL デバッガーを使用することにより、プログラムのステップ実行や変数の値を確認できます。また、Enterprise Developer に含まれているカバレッジ機能を利用すれば、通過ロジックは緑、未通過ロジックは赤と色分けされたカバレッジ結果やパーセンテージを確認できます。





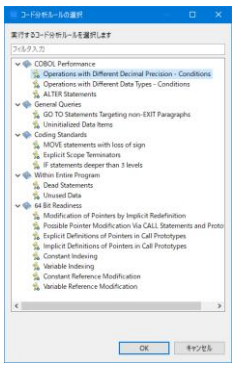


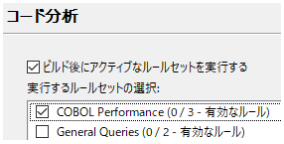
◎　クロス開発における利点

実行時の変数値をリアルタイムに確認や編集が可能で細やかなテストが実施できるため、テスト工数の削減やメインフレーム資源の使用削減に貢献します。

- デフォルトの静的解析が利用可能

製品に組み込まれている静的解析が利用可能です。コンパイル後に静的解析を行うことによりコーディングチェックを行うことができます。追加で解析クエリーを作成するには Micro Focus Enterprise Analyzer[2] が必要になります。





◎　クロス開発における利点

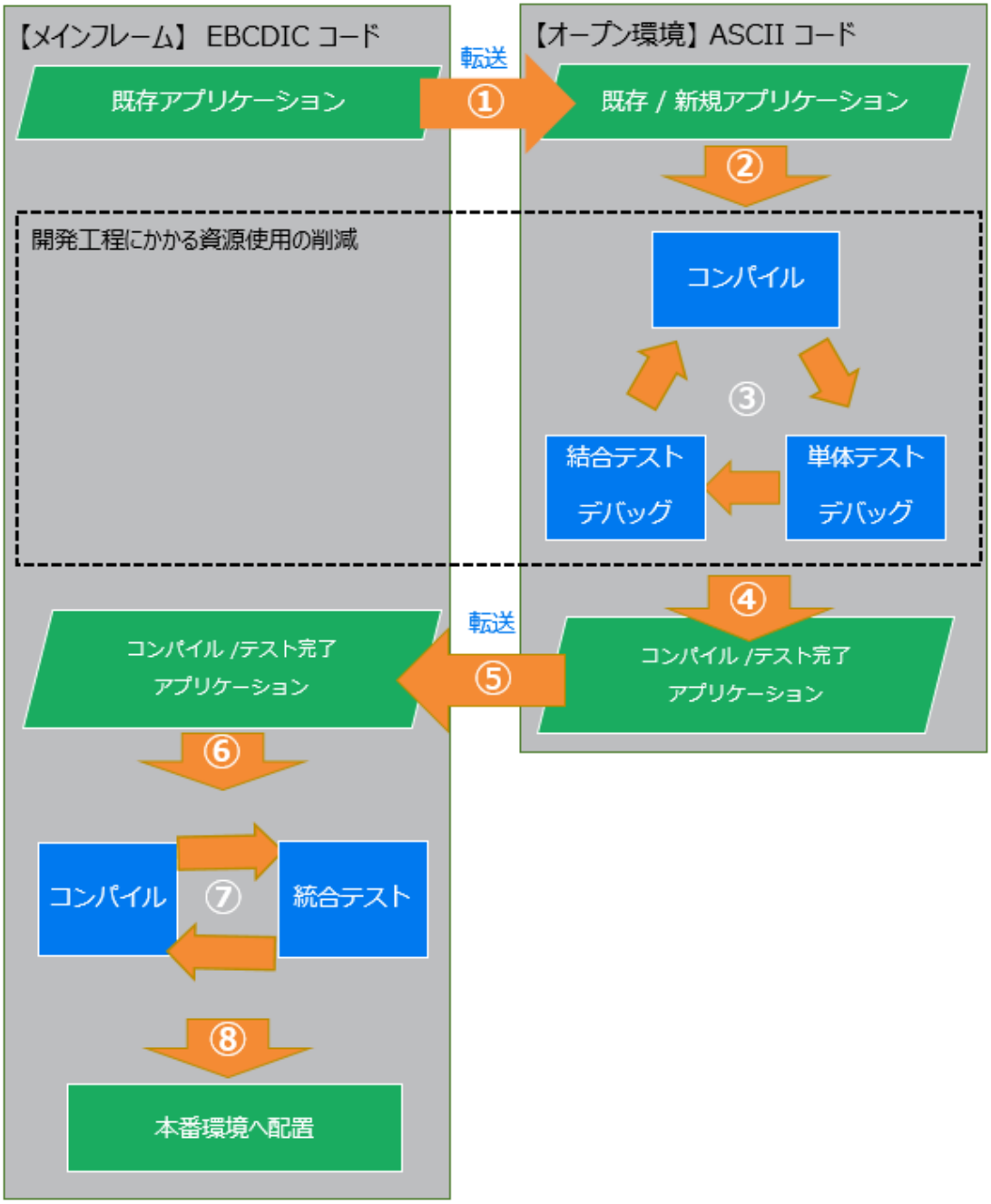
コンパイルに成功したソースが最低限のコーディング基準を守っているかなど、人の目を介さずに解析可能なため生産性の向上に貢献します。また、将来 DevOps[3] を目指す際には、ソースチェックイン→コンパイル→静的解析→デプロイなどのサイクルに組み込むことが可能になります。

[2] https://www.microfocus.co.jp/mfproducts/enterprise/analyzer/
[3] https://ja.wikipedia.org/wiki/DevOps

4. クロス開発の手順

Enterprise Developer を利用することにより、開発工程におけるメインフレーム資源の使用削減に役立ちますが、最終的には CPU が異なるためメインフレーム上での再コンパイルが必要になります。そののち、総合的なテストを完了して本番環境へ配置することとなります。

【クロス開発のフロー】

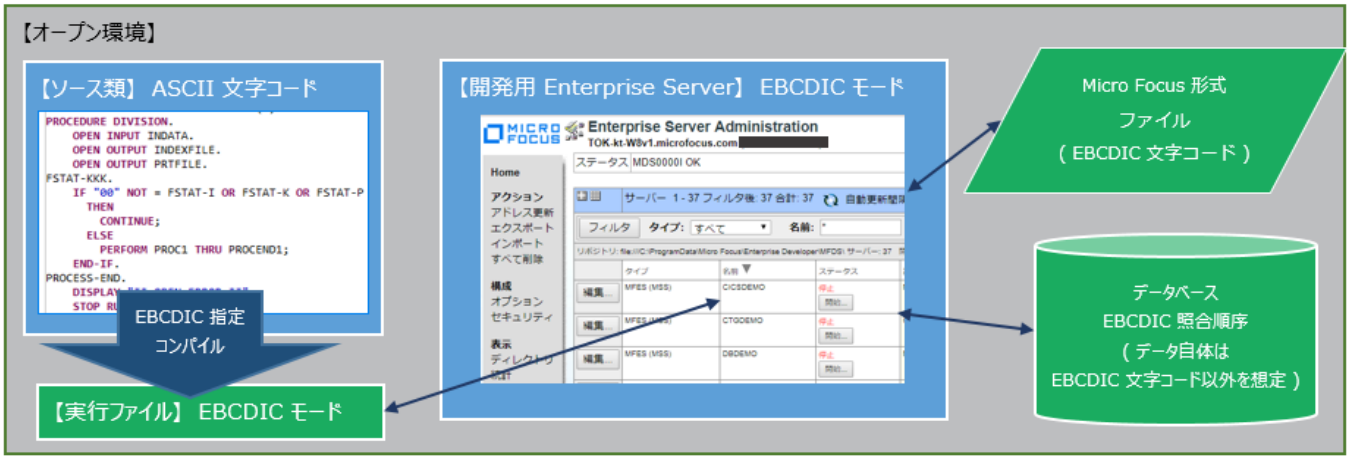


上記フローを前提に、Enterprise Developer を使用したクロス開発を具体的にどのように行うかについて、次項から技術面を含めて細かく解説します。

5. クロス開発の目的と前提

Enterprise Developer に含まれている開発用 Enterprise Server インスタンスは ASCII 文字コードデータを扱う ASCII モードと、メインフレームと同様の EBCDIC 文字コードデータを扱う EBCDIC モードが選択できます。クロス開発の場合はメインフレームとの互換性を高めるために EBCDIC モードを選択することが一般的ですが、COBOL ソースやコピー句などは、オープン環境での可視化のため ASCII 文字コードへ変換する必要があります。

【イメージ図】



テストまで実施する必要がない場合は実行解説箇所を省いてご参照ください。

6. コンパイラーの違いによる注意点と対策

メインフレームからオープン環境へ転送したアプリケーションを開発する際には下記の点に注意が必要です。注意点に該当するアプリケーションが存在し、かつ製品サポートがない場合は、修正もしくは独自のプリコンパイラーを用意するなどの対策が必要になります。

1）COBOL 構文

コンパイラーの違いによりコンパイルエラーになる代表的なものを列挙します。

- 準拠する COBOL 規格が異なる場合[4]

  製品サポート：コンパイラー指令により EXAMINE などの旧構文を使用可能にします。
- 古い COBOL コンパイラー方言を使用している場合
- COBOL 言語仕様としては不正であるがメインフレームコンパイラーが許容している場合
- 追加された予約語が利用者語と衝突している場合

  製品サポート：REMOVE コンパイラー指令により予約語から排除することが可能です。また、別の予約語に読み替える指令も用意しています。
- 特殊構文を使用している場合

  製品サポート:コンパイラー指令を指定することにより PANVALET ++INCLUDE 文などを使用可能にします。

2）実行動作

COBOL 言語仕様により結果不定と定義されている事柄について、メインフレーム実行時と Enterprise Server インスタンス実行時の挙動が異なる可能性がある代表的なものを列挙します。

- ON SIZE ERROR 句が指定されていない桁あふれの結果

  製品サポート：CHECKDIV, HOSTARITHMETIC などのコンパイラー指令により不定の結果を回避可能です。
- 初期値を設定していないデータを参照する場合

  製品サポート：INIT-BY-TYPE, DEFAULTBYTE コンパイラー指令により初期値を設定することが可能です。
- 算術結果が異なる場合

  製品サポート：Enterprise Developer では IBM 社が公開している中間結果精度の仕様をシミュレートするコンパイラー指令を用意しており、これにより互換性を保っています。例えばコンパイラー指令 ARITHMETIC"OSVS" を指定した場合は OS/VS の規則に則って切り捨てを行うなど、COBOL バージョンの指定に依存して中間結果仕様を適用しています。

採用する動作:

ENTCOBOL	Enterprise COBOL for z/OS and OS/390の規則に従い切り捨てを行う。
MF, ISO2002, ANSI	結果を切り捨ててはいけない。ARITHMETIC"MF"を指定して、できるだけ式の計算を正確に行う。切り捨てが行われない。
OS390	COBOL for OS/390の規則に従い、切り捨てを行う。
OSVS	OS/VS COBOLの規則に従い、切り捨てを行う。
VSC2	VS COBOL IIおよびCOBOL/370の規則に従い切り捨てを行う。

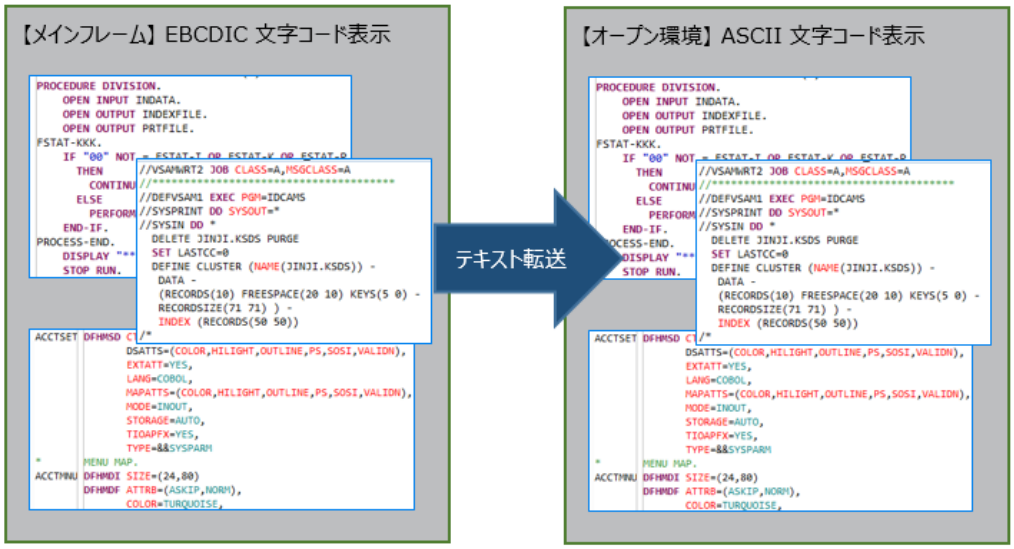
[4] 製品マニュアル）[Micro Focus Enterprise Developer] > [リファレンス] > [COBOL 言語リファレンス] > [サポートされる COBOL 言語および著作権]

7. アプリケーションの転送

クロス開発を行うために必要なアプリケーションとデータをメインフレームからオープン環境へ転送します。

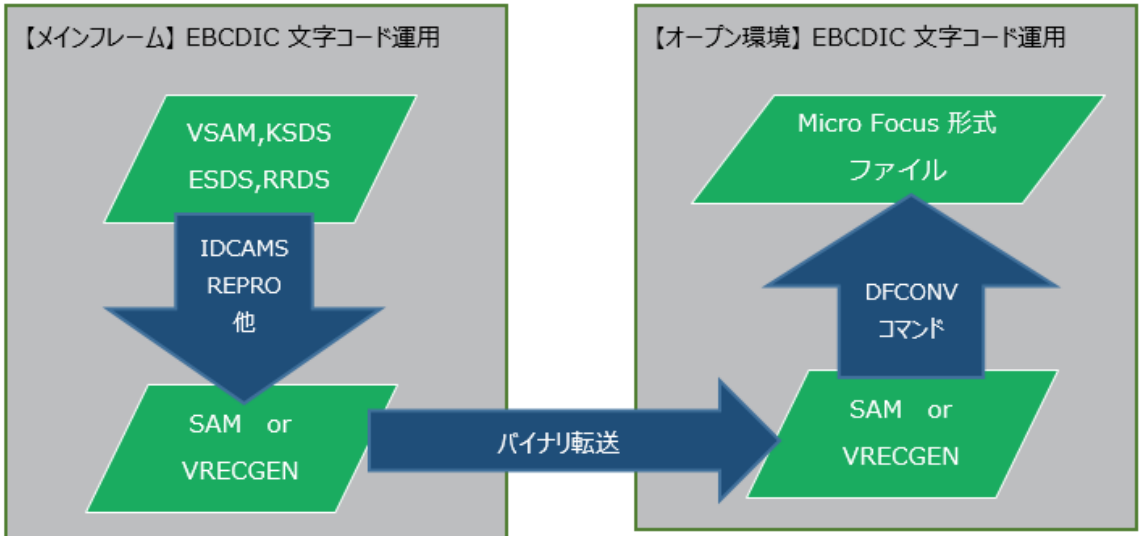
1) アプリケーション

COBOL ソース、コピー句、JCL、画面定義などのアプリケーションは ASCII 文字コードで表示や編集を行うため ASCII 文字コードへ変換後、オープン環境へテキストモードで転送します。



2) データ

データを使用したテストを計画している場合は、SAM または VRECGEN ファイル形式へ変換後、EBCDIC 文字コードのままオープン環境へバイナリモードで転送します。VRECGEN 形式ファイルに関しては製品に含まれているコマンドを利用して Micro Focus 形式のファイルへ変換する必要があります。

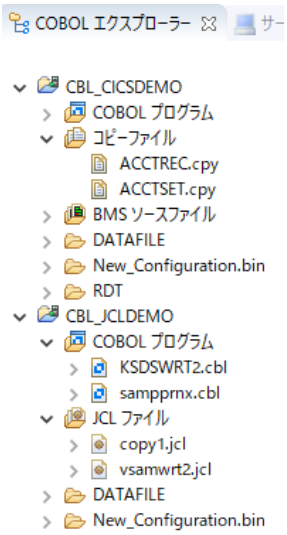


8. COBOL ソースのコンパイル

コンパイルは Eclipse IDE, Visual Studio IDE, コマンドラインから実行可能ですが、本書では Eclipse IDE を使用して、32 ビットプラットフォームをターゲットとした手順を例にあげて解説します。

1) Eclipse プロジェクトの作成

メニューから Enterprise Developer for Eclipse を起動後、メインフレーム COBOL プロジェクトを新規作成して ASCII 文字コード変換されたアプリケーションをインポートします。

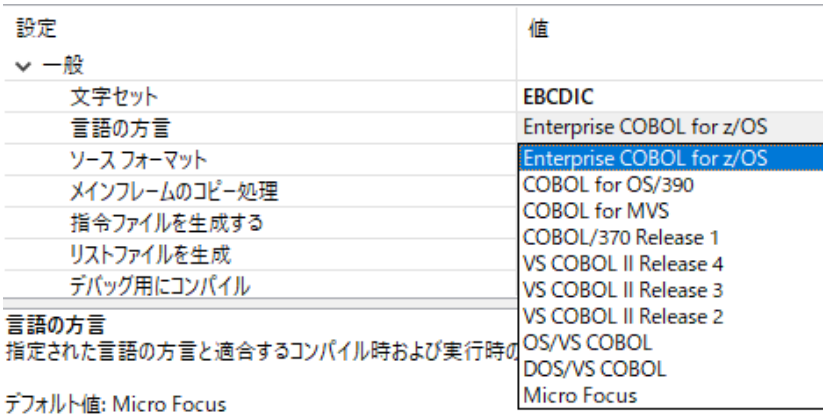


2) コンパイラー指令の設定

プロジェクトのプロパティでコンパイラー指令を指定します。
ターゲットに動的ロード形式である GNT を指定し、32 ビットプラットフォームを指定します。

変数の値、比較演算、扱うデータを EBCDIC 文字コードとするために文字セットには EBCDIC を指定します。言語方言にはメインフレームで使用している方言を指定[5]します。この指定によりメインフレームで使用しているコンパイラー指令が暗黙的に指定されます。

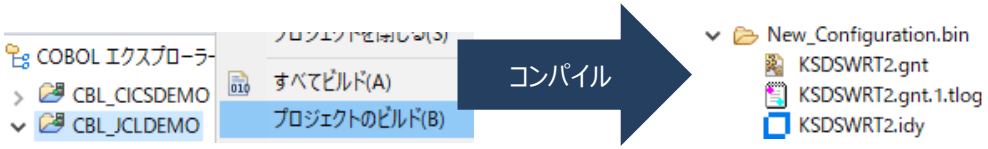
注意）方言と異なる指令がある場合は追加指令として指定する必要があります。



[5] 製品マニュアル）[Micro Focus Enterprise Developer] > [リファレンス] > [コンパイラ指令] > [Compiler Directives - Alphabetical List] > [DIALECT]

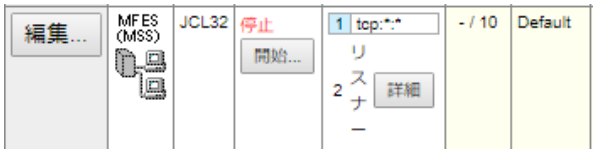
3) コンパイルの実行

プロジェクトをコンパイルして動的ロード実行ファイルを生成します。

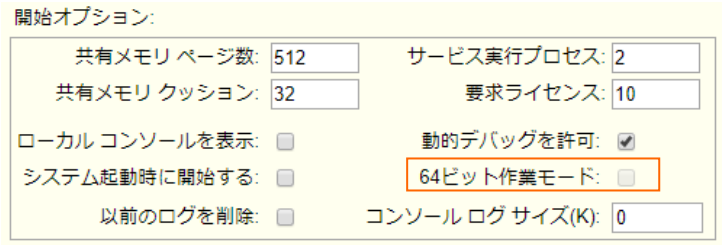


9. 開発用実行環境の設定

Enterprise Developer には開発用の実行環境である Enterprise Server を内包しています。コンパイルにより生成された 32 ビット稼働の実行ファイルを稼働させるために 32 ビット用の Enterprise Server インスタンスを作成します。



開始オプションの 64 ビット作業モードのチェックがオフの場合は 32 ビット稼働となります。



EBCDIC モードで稼働させるためには、構成情報へ環境変数 MF_CHARSET=E を設定します。この設定により、アプリケーションで扱う入出力データは EBCDIC 文字コードとして処理されます。JES 機能で出力されるジョブログはデータとして扱われないため ASCII 文字コードで生成されます。

構成情報

```
[ES-Environment]
MF_CHARSET=E
```

設定した Enterprise Server インスタンスを開始してジョブの実行を待ちます。

10. JCL の実行と結果確認

EBCDIC モードで稼働する実行ファイルと Enterprise Server インスタンスが準備できましたので、サンプル JCL を実行します。

1) サンプル JCL の内容

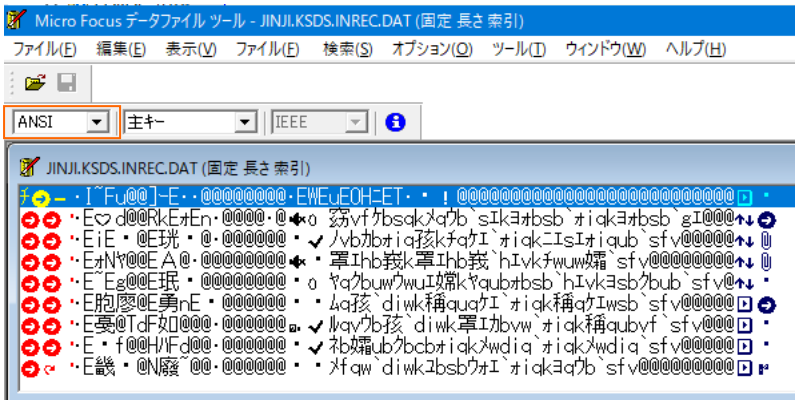
下記のサンプル JCL を実行します。カタログされている JINJI.INREC を SORTSTEP で SORT 後、JINJI.KSDS VSAM ファイルへ書き込みます。

```
//VSAMWRT3 JOB CLASS=A,MSGCLASS=A
//*************************************
//DEFVSAM1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DELETE JINJI.KSDS PURGE
  SET LASTCC=0
  DEFINE CLUSTER (NAME(JINJI.KSDS)) -
   DATA -
   (RECORDS(10) FREESPACE(20 10) KEYS(5 0) -
   RECORDSIZE(71 71) ) -
   INDEX (RECORDS(50 50))
/*
//SORTSTEP EXEC SORTD
//SORT1.SORTIN DD DSN=JINJI.INREC,DISP=(OLD,KEEP)
//SORT1.SORTOUT DD DSN=&&JINJIDAT,DISP=(NEW,PASS),
// SPACE=(800,(10,10)),DCB=(RECFM=FB,LRECL=71,DSORG=PS),UNIT=SYSDA
//SORT1.SYSIN DD *
  SORT FIELDS=(1,5,CH,A)

//APPL1 EXEC PGM=KSDSWRT2
//SYSOUT DD SYSOUT=*
//KSDSFILE DD DSN=JINJI.KSDS,DISP=SHR
//INDD DD DSN=&&JINJIDAT,DISP=(OLD,DELETE)
//PRINTER DD SYSOUT=*
//*************************************
//VERIFY1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  REPRO INDATASET(JINJI.KSDS) -
  OUTFILE(SYSPRINT)
/*
```

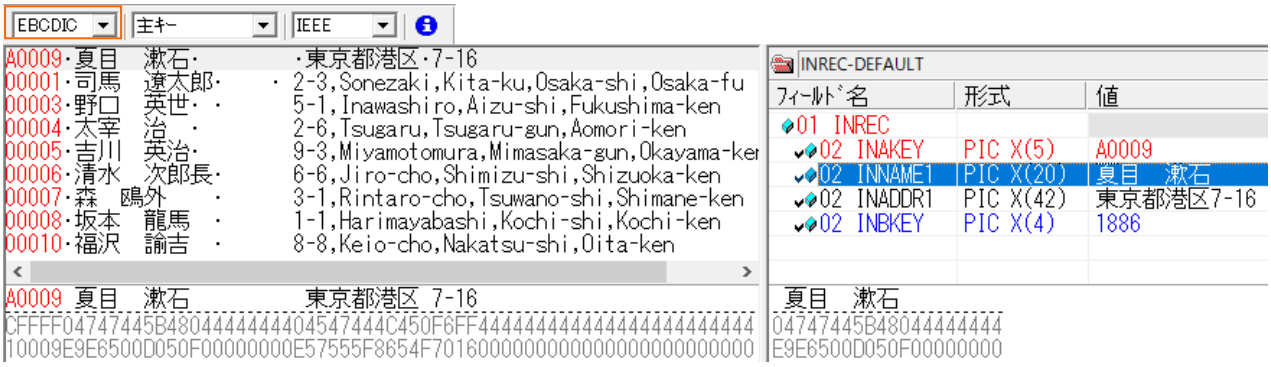
2) 入力データの内容

カタログされている JINJI.INREC ファイルの内容を Enterprise Developer に含まれているデータツールで確認します。

EBCDIC 文字コードデータのため、ASCII 文字コード表示では文字化けして読むことができません。



EBCDIC 文字コード表示へ切り替えて内容を表示すると可視化され、HEX 表示をすることで EBCDIC 文字コードが格納されていることが確認できます。漢字の前後にはシフトコードも確認できます。



3) プログラムの内容

サンプル JCL から呼ばれるプログラムの内容を確認します。確認のため IF 文で HEX コード比較と大小比較を加え、キーが 00007 のレコードは除外するよう記述してあります。

```
READ INDATA
  AT END
    SET LOOP1 TO TRUE
  NOT AT END
    IF INAKEY = X"F0F0F0F0F1" THEN
       DISPLAY "THIS KEY IS EBCDIC = " INAKEY
    END-IF
    IF INAKEY < X"F0F0F0F0F1" THEN
       DISPLAY "THIS KEY IS SMALLER THAN 00001 = " INAKEY
    END-IF
    IF INAKEY NOT = X"F0F0F0F0F7" THEN
      MOVE INREC TO KREC
      MOVE INREC TO PREC
      WRITE PREC
      WRITE KREC
        INVALID KEY DISPLAY "INVALID"
        NOT INVALID KEY CONTINUE
      END-WRITE
    END-IF
END-READ
```

4) 結果の確認

JCL を実行して結果を確認してみます。

- IF 文による DISPLAY 出力結果

```
THIS KEY IS SMALLER THAN 00001 = A0009
THIS KEY IS EBCDIC = 00001
```

とあり、EBCDIC 文字コードでの比較が行われていること、EBCDIC 照合順序で大小比較が行われていることが確認できます。

- カタログされた出力データファイルの内容

Enterprise Server インスタンスの管理画面から、カタログされた DS NAME をクリックするとコンバート機能が動作するためデータが可視化されていますが、実ファイルをテキストエディターで表示すると文字化けして読むことができません。

キー項目が 00007 は出力しないようコーディングしているため該当レコードは出力されていません。

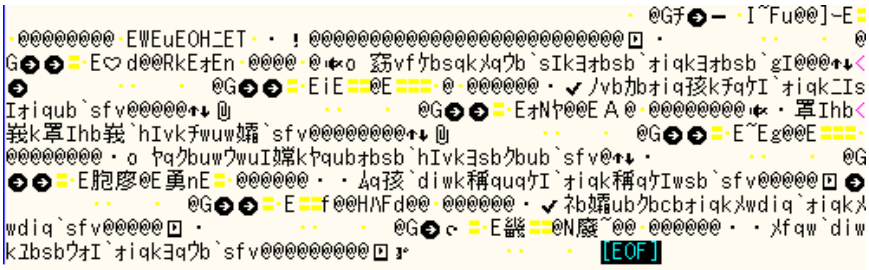
【Enterprise Server インスタンス管理画面から表示したデータ】

CATALOG Entry　Refresh

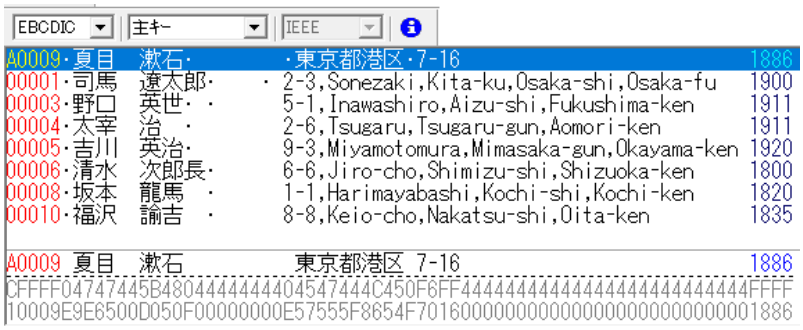
Content-Type: text/plain

```
A0009夏目　漱石　　東京都港区7-16                          1886
00001司馬　遼太郎　2-3,Sonezaki,Kita-ku,Osaka-shi,Osaka-fu  1900
00003野口　英世　　5-1,Inawashiro,Aizu-shi,Fukushima-ken    1911
00004太宰　治　　2-6,Tsugaru,Tsugaru-gun,Aomori-ken        1911
00005吉川　英治　　9-3,Miyamotomura,Mimasaka-gun,Okayama-ken 1920
00006清水　次郎長　6-6,Jiro-cho,Shimizu-shi,Shizuoka-ken    1800
00008坂本　龍馬　　1-1,Harimayabashi,Kochi-shi,Kochi-ken    1820
00010福沢　諭吉　　8-8,Keio-cho,Nakatsu-shi,Oita-ken        1835
```

【テキストエディターから表示したデータ】



再度、データツールの EBCDIC 文字コード表示にて確認すると正しく出力されていることがわかります。



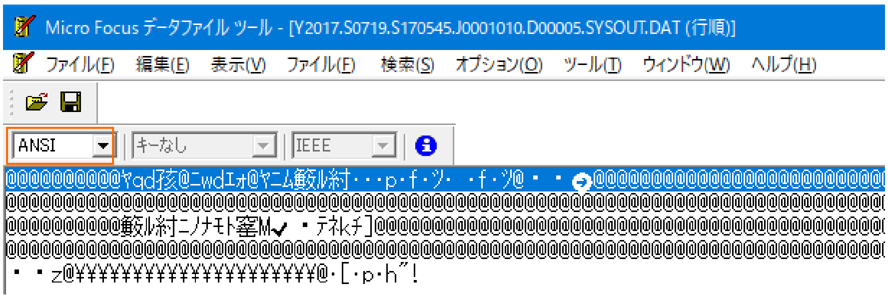
- ## ジョブログの内容

ASCII 文字コードで出力されるため、テキストエディターへ表示しても読むことができます。

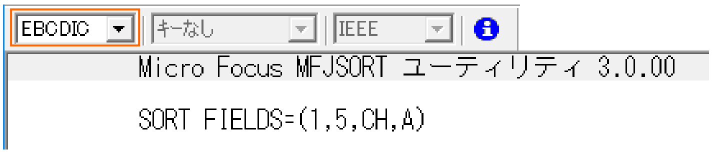
```
   *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
   *-*-*  Micro Focus ESJCL  EBCDIC JES2 Version ED3.0_022                                    *-*-*
   *-*-*  Copyright (C) 1997-2017 Micro Focus. All rights reserved.                           *-*-*
   *-*-*  Job: 0001014 Name: VSAMWRT3 User: JESUSER  Date: 07/20/17 Time: 13:07:03            *-*-*
   *-*-*  File: $TXRFDIR/T000000062.T                                                         *-*-*
   *-*-*  DSN:                                                                                *-*-*
   *-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

  1 //VSAMWRT3 JOB CLASS=A,MSGCLASS=A
  2 //*************************************
  3 //DEFVSAM1 EXEC PGM=IDCAMS
  4 //SYSPRINT DD SYSOUT=*
  5 //SYSIN DD *
 14 //SORTSTEP EXEC SORTD
    XXSORTD  PROC
    XXSORT1  EXEC  PGM=SORT
 15 //SORT1.SORTIN DD DSN=JINJI.INREC,DISP=(OLD,KEEP)
 16 //SORT1.SORTOUT DD DSN=&&JINJIDAT,DISP=(NEW,PASS),
 17 // SPACE=(800,(10,10)),DCB=(RECFM=FB,LRECL=71,DSORG=PS),UNIT=SYSDA
 18 //SORT1.SYSIN DD *
 21 //APPL1 EXEC PGM=KSDSWRT2
 22 //SYSOUT DD SYSOUT=*
 23 //KSDSFILE DD DSN=JINJI.KSDS,DISP=SHR
 24 //INDD DD DSN=&&JINJIDAT,DISP=(OLD,DELETE)
 25 //PRINTER DD SYSOUT=*
 26 //***************************************
 27 //VERIFY1 EXEC PGM=IDCAMS
 28 //SYSPRINT DD SYSOUT=*
 29 //SYSIN DD *
 33 //SYSIN DD * ***MFE JCL GENERATED STMT*** For more info, see JCLCM0094I
*** JCLCM0180I Job ready for execution.
*** Execution on Server JCL32     Process      5972
```

- ## SYSPRINT, SYSOUT の内容

データツールから確認してみます。EBCDIC 文字コードで出力されるため ASCII 文字コード表示では文字化けしています。



データツール内の文字コードを EBCDIC 文字コード表示へ変更すると可視化されます。



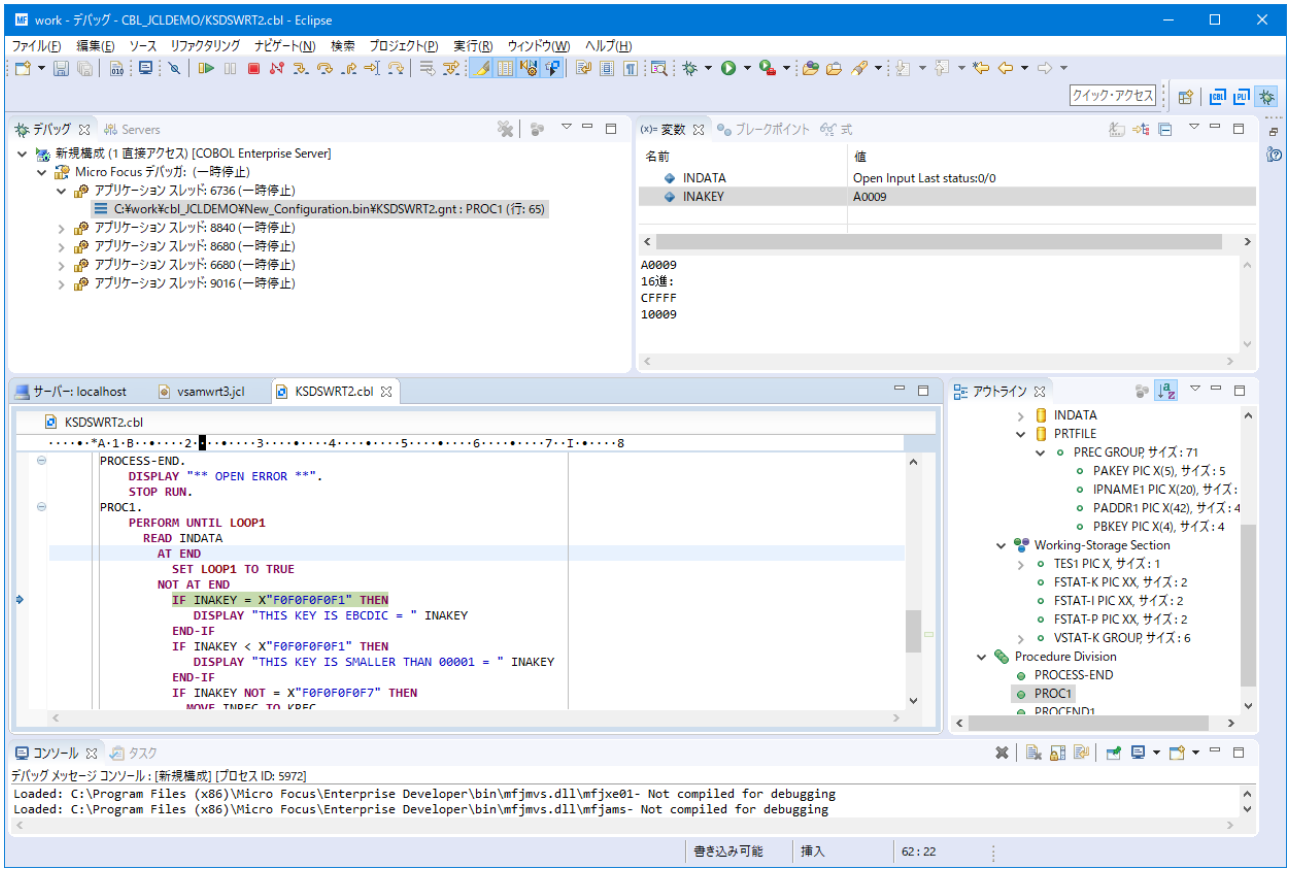
- ## 結論

EBCDIC 文字コードデータの読み込み、EBCDIC 文字コードデータの出力が確認でき、文字コードに依存したコーディング箇所の修正は必要ないことが確認できました。また、ISAM ファイルに関しては Micro Focus 形式にするコマンドを実行する必要はありますが、データを EBCDIC 文字コードのまま使用できることも確認できました。

11. 生産性の向上

前項ではプログラムにコーディングされた DISPLAY 命令で出力されたデータ内容を確認しましたが、Enterprise Developer では COBOL プログラムのステップ実行が可能です。実際にデバッガーからデータ値を確認します。

Eclipse IDE でデバック実行を選択した後にサンプル JCL を実行すると、デバッグパースペクティブというデバッグ用の画面が表示され、ステップ実行によりその時点の変数値を確認できます。右上の変数タブでは HEX 表示により 値が EBCDIC コードであることが確認できます。



このようなデバッグ機能を利用することで、どのステップでどのような値が格納されているのか、加えて、値には ASCII 文字コードが格納されているのか EBCDIC 文字コードが格納されているのかをリアルタイムに確認できるため、テスト工数の削減に役立ちます。
また、COBOL 構文、JCL 構文についても入力値の正当性をチェックする機能が搭載されていることから、コーディングミスをリアルタイムに察知することができ、生産性の向上が望めます。

12.データベースの EBCDIC 照合指定

データベースと連携して EBCDIC モードでアプリケーションを稼働させる際に問題点として挙げられるのは照合順序です。たとえば SQL 文で ORDER BY を指定した場合、データベース側の設定が ASCII 文字コードであれば EBCDIC 文字コード順には並びません。また WHERE 句で大小比較を行っている場合には意図したレコードを取得することができません。日本語 EBCDIC 文字コード照合指定を実現するには Micro Focus 製品からではなく、データベースの API などを使用して照合順序を設定する必要があります。例として DB2 を使用した方法を解説します。

1) DB2 提供のサンプルコピー

インストールした DB2 のパスに COBOL 向けのサンプルプログラムとコピー句が含まれています。これらのフォルダを操作可能なパスへコピーします。

パス例)

プログラム：C:¥Program Files¥IBM¥SQLLIB¥samples¥cobol_mf

コピー句：C:¥Program Files¥IBM¥SQLLIB¥include¥cobol_mf

2) サンプルプログラムの修正

フォルダに含まれている ebcdicdb.cbl をテキストエディターで編集します。

- 使用するコピー句の変更

【編集前】

```
*--> sqlb0x67.cobol
 copy "sqle850b.cbl".
```

【編集後】

```
*--> sqlb0x67.cobol
 copy "sqle932a.cbl".
```

sqle932a.cbl はサンプルコピー句に含まれているもので コメントに書かれているコードセットを使用します。

```
Function = Include File defining:
             Collating Sequence
               Source: CODE PAGE 932 (ASCII Japanese)
               Target: CCSID 5035 (EBCDIC Japanese)
```

- 使用する名前の指定

【編集後】

```
* Variables for Create/Drop database
 77 DBNAME                pic x(10)  value "db2demo".
 77 DBNAME-LEN            pic s9(4) comp-5 value 7.
 77 ALIAS                 pic x(10)  value "db2demo".
 77 ALIAS-LEN             pic s9(4) comp-5 value 7.
```

任意の DBNAME と ALIAS を指定して、その長さを LEN へ指定します。

- コードセット、国情報を変更

【編集前】

```
* setup database description block SQLEDBDESC
     move SQLE-DBDESC-2  to SQLDBDID.
     move 0              to SQLDBCCP.
     move SQL-CS-USER    to SQLDBCSS.
     move SQLE-850-037   to SQLDBUDC.
     move "EBCDIC"       to SQLDBCMT.
     move 0              to SQLDBSGP.
     move 10             to SQLDBNSG.
     move -1             to SQLTSEXT.

     SET SQLCATTS        TO NULLS.
     SET SQLUSRTS        TO NULLS.
     SET SQLTMPTS        TO NULLS.

* setup database country information
* structure SQLEDBCOUNTRYINFO
     move "IBM-850" to SQLDBCODESET of SQLEDBCOUNTRYINFO.
     move "En_US"   to SQLDBLOCALE of SQLEDBCOUNTRYINFO.
```

【編集後】

```
* setup database description block SQLEDBDESC
     move SQLE-DBDESC-2  to SQLDBDID.
     move 0              to SQLDBCCP.
     move SQL-CS-USER    to SQLDBCSS.
     move SQLE-932-5035  to SQLDBUDC.
     move "EBCDIC"       to SQLDBCMT.
     move 0              to SQLDBSGP.
     move 10             to SQLDBNSG.
     move -1             to SQLTSEXT.

     SET SQLCATTS        TO NULLS.
     SET SQLUSRTS        TO NULLS.
     SET SQLTMPTS        TO NULLS.

* setup database country information
* structure SQLEDBCOUNTRYINFO
     move "IBM-932" to SQLDBCODESET of SQLEDBCOUNTRYINFO.
     move "Ja_JP"   to SQLDBLOCALE of SQLEDBCOUNTRYINFO.
```

- DROP 以降をコメント化

【編集後】

```
*    display "DROPping the database DBEBCDIC".
*-->
*****************************
* DROP DATABASE API called *
*****************************
*    call DB2API "sqlgdrpd" using
*                          by value     reserved1
*                          by value     DBNAME-LEN
*                          by reference sqlca
*                          by value     reserved2
*                          by reference DBNAME
*                     returning rc.
*<--
*
*    move "dropping the database" to errloc.
*    call "checkerr" using SQLCA errloc.

 end-ebcdicdb. stop run.
```

DROP を行わない場合はコメント化します。

3) サンプルプログラムのコンパイル

使用ビット数に沿ったコマンドプロンプトからコンパイルを実行します。

① フォルダに含まれている checker.cbl をコンパイルします。

コンパイルコマンド）cobol checkerr.cbl;

② 日本語 EBCDIC 照合指定に変更したプログラムをコンパイルします。

コンパイルコマンド）cobol ebcdicdb.cbl;

③ コンパイルした上記プログラムを DB2 の LIB とリンクして実行ファイルを作成します。

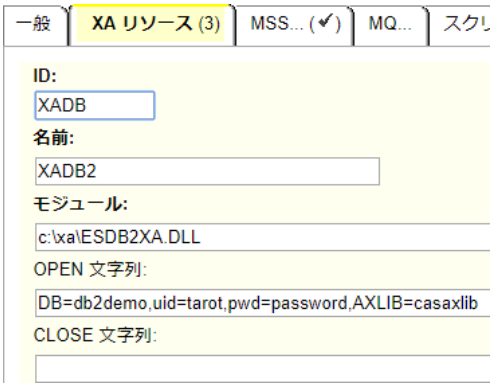
リンクコマンド）cbllink -l ebcdicdb.obj checkerr.obj db2api.lib;

④ 作成した実行ファイルを実行します

コマンド）ebcdicdb;

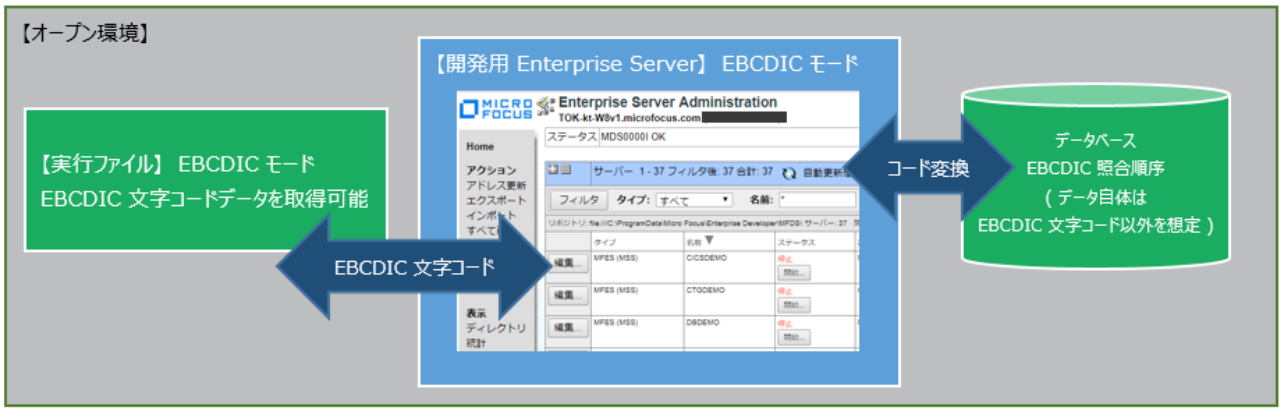
上記手順により、日本語 EBCDIC 照合指定の DB2 データベースが作成されます。

Enterprise Server インスタンスに登録する XA スイッチモジュールを介して、このデータベースへ接続します。



4) Enterprise Server インスタンスとの連携

前項で作成したデータベースと連携する際は Enterprise Server インスタンス内部でコード変換が行われますので、COBOL プログラムから見ると FETCH されたデータは EBCDIC 文字コードとなります。



13.メインフレームのコンパイル結果比較

Enterprise Developer でコンパイルした同じソースをメインフレームでコンパイルして結果を確認します。

1) 対象 COBOL ソース

コンパイルエラーが発生するように、文字列を 9 タイプ項目へ転記しています。

```
01 SMP-9 PIC 9(05) VALUE ZERO.

PROCEDURE DIVISION.
    MOVE "AAAAA" TO SMP-9.
```

2) Enterprise Developer コンパイル結果

レベル E のコンパイルエラーが発生しており、実行ファイルを生成しています。

```
*
* エラー
* - 行 - 桁  コード/重大度 説明
*     49   33 1026-E        送り出し側が文字定数である - 代わりにゼロを転記する
* Micro Focus COBOL                V3.0 revision 000 Compiler
* Copyright (C) Micro Focus 1984-2017. All rights reserved.
*                                                    REF GKR-080300000A0
* ページの最終メッセージ:    6
* 合計メッセージ:     1
* 回復不能エラー:     0                  重大な警告:     0
* エラー       :     1                  警告       :     0
* 備考 :     0                         フラグ     :     0
* データ:        4208     コード:        3028
```

3) メインフレームのコンパイル結果

同じ箇所でレベル S のコンパイルエラーが発生しており、実行ファイルは生成されていません。

また、インフォメーションとして RECORDING MODE 指定がないので “F” と想定する、と出力されています。

```
 SDSF OUTPUT DISPLAY MFIMKIBK JOB00454  DSID   101 LINE  NOT PAGE MODE DATA
 COMMAND INPUT ===> _                                    SCROLL ===> PAGE
******************************** TOP OF DATA ********************************
PP 5648-A25 IBM COBOL for OS/390 & VM  2.1.1  in progress ...
LineID  Message code  Message text
    23  IGYGR1216-I   A "RECORDING MODE" of "F" was assumed for file "INDATA".
    29  IGYGR1216-I   A "RECORDING MODE" of "F" was assumed for file "PRTFILE".
    49  IGYPA3005-S   ""AAAAA"" and "SMP-9 (NUMERIC INTEGER)" did not follow
                      the "MOVE" statement compatibility rules.  The statement
                      was discarded.
Messages    Total    Informational    Warning    Error    Severe    Terminating
Printed:        3              2                                1
End of compilation 1,  program KSDSWRT2,  highest severity 12.
Return code 12
****************************** BOTTOM OF DATA ******************************
```

4) コンパイル結果の差異について

クロス開発において、オープン環境で出来るだけ多くのコンパイルエラーを検出して修正するという目的から、Enterprise Developer ではエラーとなる箇所を多く検出するよう前項で示したようなレベルの差異となっています。これによりコンパイルが中断されることなく多くのエラー箇所をマーキングでき、メインフレーム資源を使用する前に多くのエラーを認識し修正することが可能になります。

14. おわりに

Enterprise Developer を使用したクロス開発を実施することにより、メインフレームの資源を一切使用することなくオープン環境でコンパイルチェックとテスト、デバッグを行うことができ、メインフレーム資源使用にかかるコストの削減やテストによる品質向上が望めます。

また、開発用 Enterprise Server インスタンス を使用すれば、個々のユーザが「疑似メインフレーム」を占有する感覚で開発が可能になり、これによる開発工数やテスト工数の削減など、生産性の向上も期待できます。さらにオープン環境に存在する様々なツールとの連携が可能となります。

開発環境においても Java 開発者や .NET 開発者が利用している Eclipse IDE, Visual Studio IDE を使用することにより、世代を超えた技術の継承も現実的になります。COBOL 言語に偏見がある世代も Micro Focus COBOL は JVM 言語[6]のひとつであると知っていただける良い機会と考えています。

補足：稼働環境

本書は下記環境で実施されました。

1) OS

Windows 10 Enterprise

2) プロセッサ

Intel® Core™ i7-8700 CPU @ 3.20HGz 3.19 GHz

3) システムの種類

64 ビットオペレーティング システム x64 ベース プロセッサ

4) Micro Focus 製品 バージョン

Micro Focus Enterprise Developer 5.0J

注意）Micro Focus Enterprise Server 5.0J と同等の開発用実行環境を含んでいます。

6 https://en.wikipedia.org/wiki/List_of_JVM_languages