Rocket software

Enterprise Developer を使用したクロス開発

Version.3
Last updated: 2026 年 3 月

[Subject]

目次

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.amc.rocketsoftware.co.jp/support/manuals.asp

1. はじめに

IBM メインフレームは、現在も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、多くの人的資源や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。

Enterprise Developer に内蔵されている COBOL コンパイラは、IBM メインフレームのクロス開発で約半世紀にわたって利用されてきた実績があります。各開発者がオープン環境で Enterprise Developer の機能を利用することにより、COBOL アプリケーションの開発や保守によるランニングコストの削減が望めます。

機能例）

- EBCDIC 文字コードデータをオープン環境で利用できる
- IBM COBOL コンパイラの各種バージョンに対して個別に高い互換性を持つ
- 開発用実行環境の JCL、CICS、IMS DB/DC 互換機能を利用したテストができる

2. 問題点の把握

IBM メインフレームを使用した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。

1）IBM 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。

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、コストが増加する。

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。

4）次世代への継承

IBM 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、IBM 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、移行先環境は何が適切なのか、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。

Enterprise Developer は、過去の投資を無駄にせず、実績のある COBOL アプリケーションだけではなく、PL/I アプリケーションもオープン環境で稼働できる開発環境製品です。

3. 解決策の選択肢

現状の問題点に対して、一般的な解決策は以下の通りです。

1)IBM メインフレームに新たな区画や資源を確保

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、全問題点の解決は望めない。

2)リプラットフォーム

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。移行計画や移行設計、新旧比較テストなど一定の時間が必要になるが、将来に向けて段階的なモダナイゼーションも視野に入れることができ、全問題点の解決が望める。

3)リライト

現在のアプリケーションを別言語に書き換える方法。現行の仕様把握から新機能の設計、全量テストなど、多大な時間が必要になる。自社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、全問題点の解決が望める。

4)クロス開発

開発と単体テストはオープン環境で実施し、IBM 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。クロス開発手法が浸透するまでの時間が必要になるが、リプラットフォームやリライトと比較すれば最小限に抑えられる。オープン環境で他開発言語と同じ統合開発環境を利用すれば、次世代への継承が見込め、オープン化への試行としても役立ち、問題点の解決が望める。

本書では、コストを削減し、かつ将来を見据えたクロス開発手法に着目し、これを解説します。

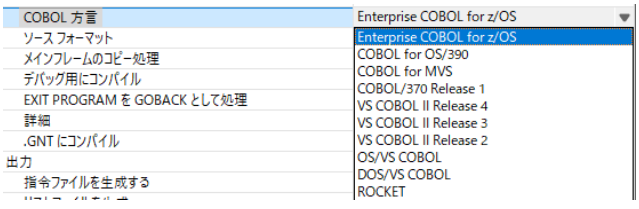
4. Enterprise Developer で実現できること

Enterprise Developer は IBM メインフレームの JCL、CICS、IMS DB/DC 互換機能を持ち、リプラットフォームも可能な製品です。また、製品のコンパイラ技術により、COBOL ソースから JVM や.NET 上で稼働可能な Java バイトコードや CIL コードの実行モジュールを生成することもでき、将来のオープン化に向けた具体的なイメージを持つこともできます。

クロス開発においては、以下の機能を利用することができます。

1)IBM メインフレームのコンパイラバージョンが指定可能

コンパイル時に、IBM メインフレームのコンパイラバージョンを方言として指定できます。これにより方言に沿った多数の指令が暗黙的に指定され、IBM メインフレームとの高い互換性を保つことができます。

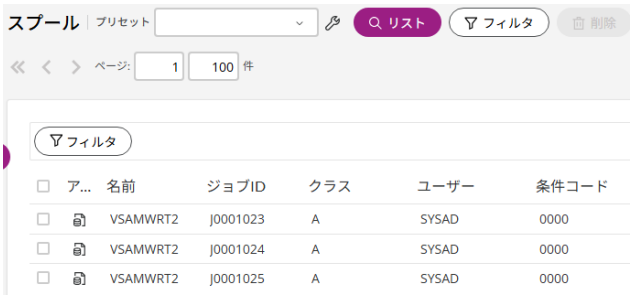


◎クロス開発における利点

オープン環境でコーディング、コンパイルし、エラーを検出、修正することができるため、IBM メインフレーム資源の使用削減に貢献します。

2）IBM メインフレームの JCL が利用可能

JCL 互換機能により IBM 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

疑似 IBM メインフレーム環境を占有しながら JCL を実行し、テストができるため、開発工数の削減や IBM メインフレーム資源の使用削減に貢献します。

3）CICS 構文が利用可能

CICS 互換機能により、EXEC CICS 構文や、一般的に使用される API、SPI コマンドの大部分をオープン環境で実行できます。TN3270 エミュレータを、開発用実行環境である 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、BMS 画面定義に沿った画面が表示されます。PCT などの、CICS リソース定義ファイル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 CICS へのログインやオンライン処理のテストが実施できるため、IBM メインフレーム資源の使用削減に貢献します。

4）IMS DB/DC が利用可能

IMS 互換機能により IMS 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。MFS 画面定義や MPP、BMP、DLI もサポートしています。ただし、データファイルは製品形式のファイルへ変換する必要があります。

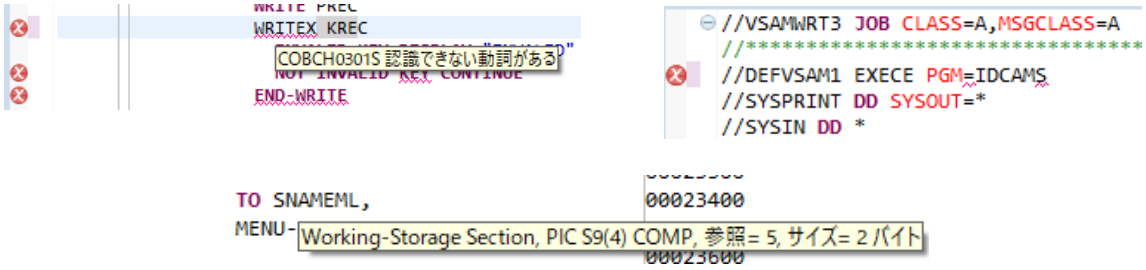
```
PROCEDURE DIVISION   USING PCB-LT
                           ALT-PCB
                           DEMO-PCB.
    ENTRY 'DLITCBL'  USING PCB-LT
                           ALT-PCB
                           DEMO-PCB.

    CALL 'CBLTDLI' USING
        ISRT
        PCB-LT
        DC001TIO
        OTDEMO91.
```

◎クロス開発における利点

CBLTDLI、EXEC DLI 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検出し修正できます。
オープン環境でテストが実施できるため、IBM メインフレーム資源の使用削減に貢献します。

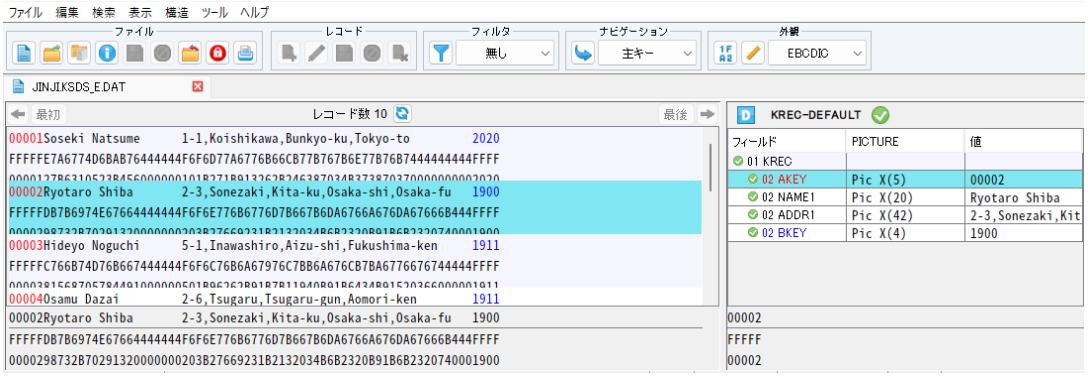
5）開発支援機能の利用が可能

COBOL エディター、JCL エディターを使用することにより、リアルタイムにコーディング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、製品に同梱されているデータファイル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。



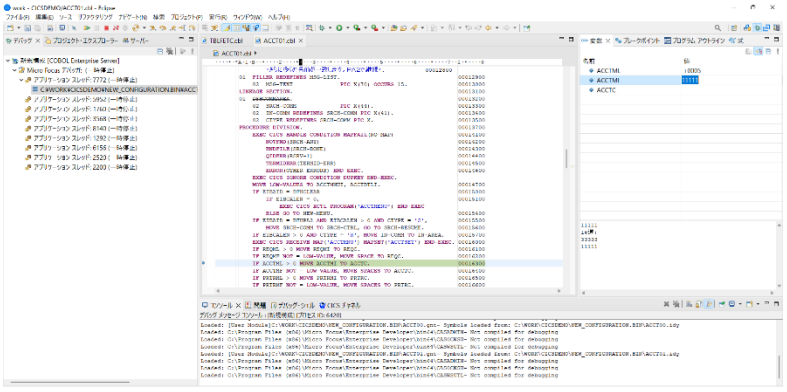
◎クロス開発における利点

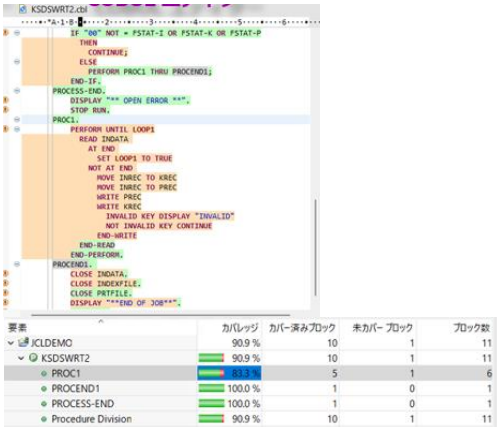
コーディングエラーやタイプミスを未然に防ぐことができるため、開発工数の削減に貢献します。テストで使用する EBCDIC 文字コードデータを確認できるため、IBM メインフレーム資源の使用削減に貢献します。



6)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。



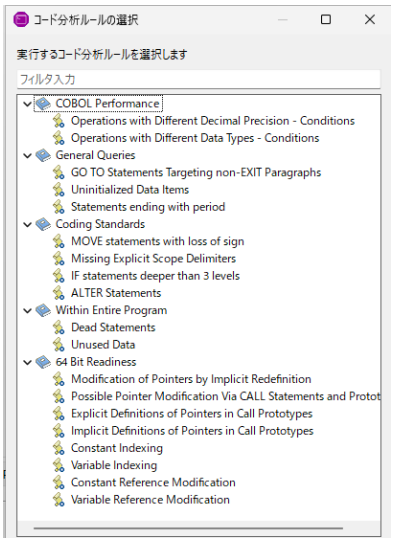


◎クロス開発における利点

実行時の変数値をリアルタイムに確認できるため、テスト工数の削減や IBM メインフレーム資源の使用削減に貢献します。

7）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でき、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。Enterprise Analyzer を利用すれば、独自の解析クエリーを作成し、連携することができます。

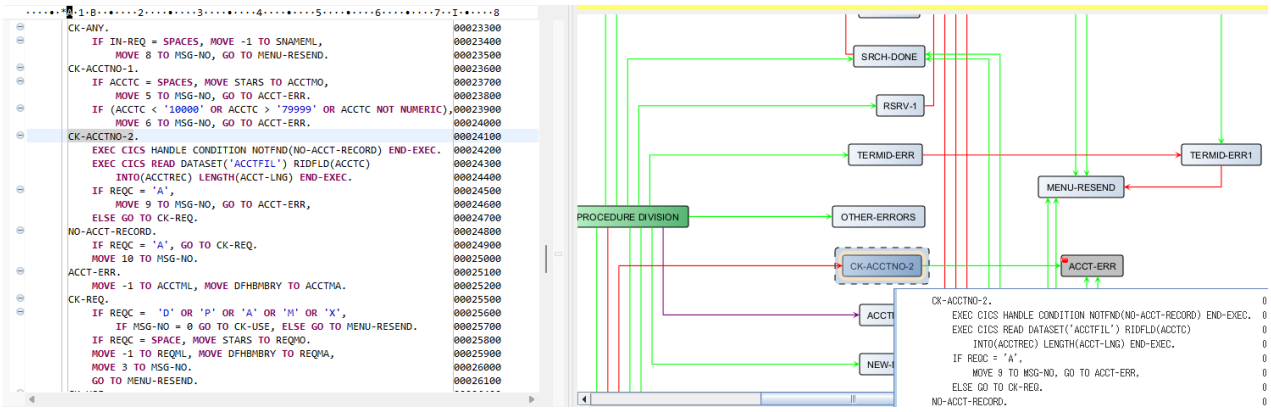


◎クロス開発における利点

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、品質向上に貢献します。また、将来 DevOps 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です。

8）プログラムフローグラフが利用可能

複雑なプログラムの分析や可視化を支援する機能です。フロー描画のセクションや矢印をダブルクリックすると、ソースコードと連動して該当コードに位置づけられます。また、マウスオーバーにより該当コードをツールチップで確認することもできます。



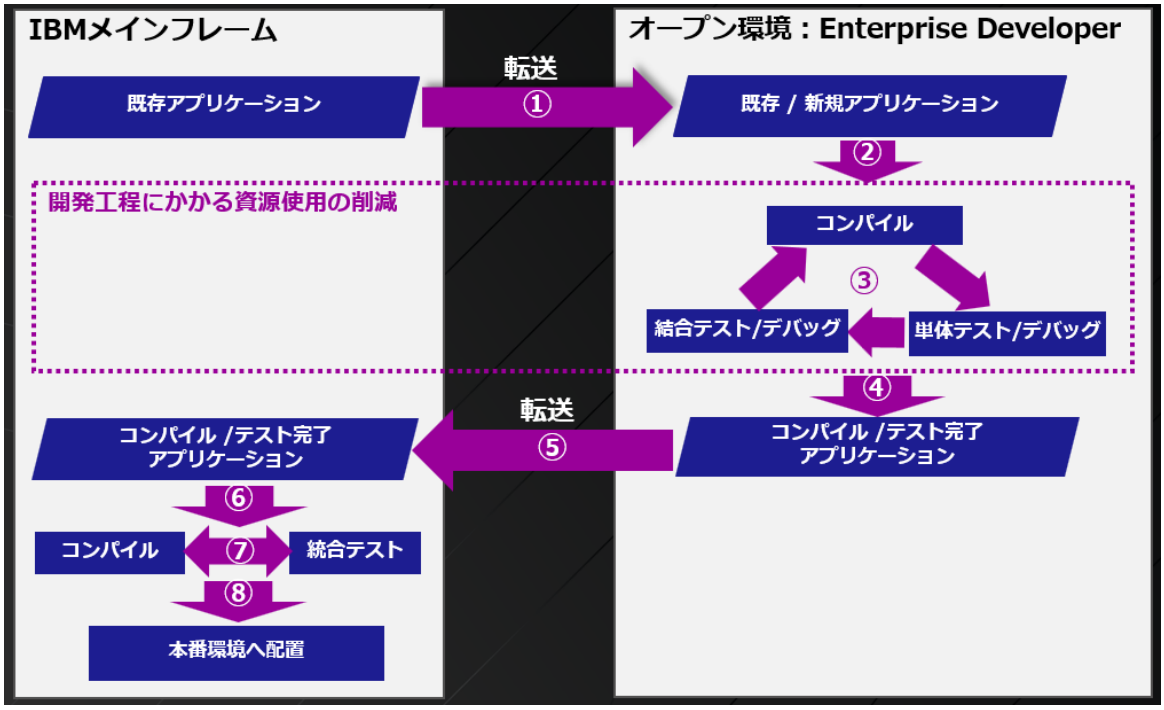
◎クロス開発における利点

COBOL プログラムの仕様書がない場合でも、内容を把握しながらメンテナンスすることができます。

5. 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 IBM メインフレーム資源の使用削減に役立ちますが、総合テストに向けて IBM メインフレーム上での再コンパイルが必要になります。

【クロス開発のフロー】

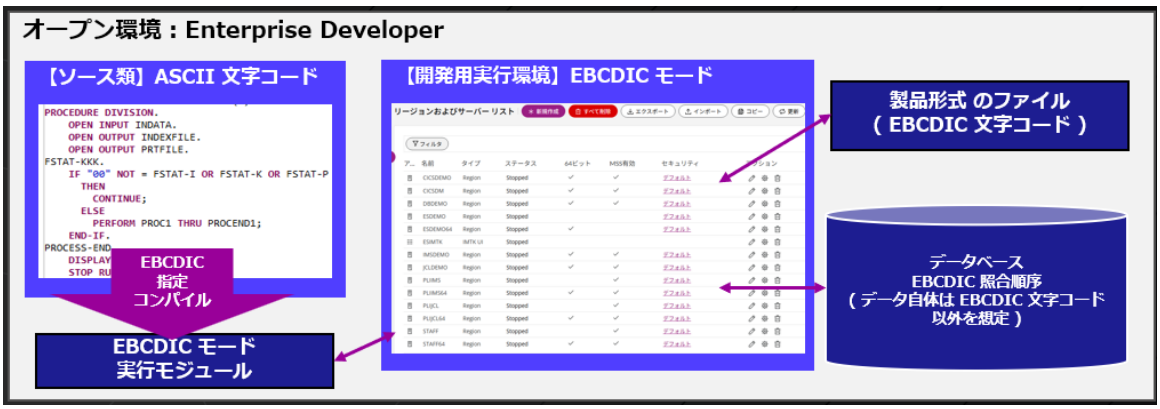


上記フローを前提に、Enterprise Developer を使用したクロス開発を具体的にどのように行うかについて、次項から技術面を含めて細かく解説します。

6. クロス開発の目的と前提

Enterprise Developer に含まれている開発用実行環境では、ASCII/SJIS 文字コードデータを扱う ASCII モードと、IBM 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 IBM 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。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/SJIS 文字コードへ変換する必要があります。

【イメージ図】



7. コンパイラの違いによる注意点と対策

IBM 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
  製品サポート： EXAMINE などの旧構文をコンパイラ指令により使用可能にします。
- 古い COBOL コンパイラ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、IBM 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：PANVALET ++INCLUDE 文などをコンパイラ指令により使用可能にします。

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、IBM メインフレーム実行時と開発用実行環境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に互換性を持つコンパイラ指令を用意しています。例えばARITHMETIC”OSVS”コンパイラ指令を指定した場合は 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。

コンパイラ指令の例）

パラメーター：

arith-type

適用する動作：

ENTCOBOL

Enterprise COBOL for z/OS および OS/390 の規則に従って切り捨てを行います。

MF、ISO2002、ANSI

中間結果を切り捨てません。ARITHMETIC"MF"、ARITHMETIC"ISO2002"、または ARITHMETIC"ANSI" を指定した場合、切り捨てが行われないため、式の計算が最も正確になります。

OS390

COBOL for OS/390 の規則に従って切り捨てを行います。

OSVS

OS/VS COBOL の規則に従って切り捨てを行います。

VSC2

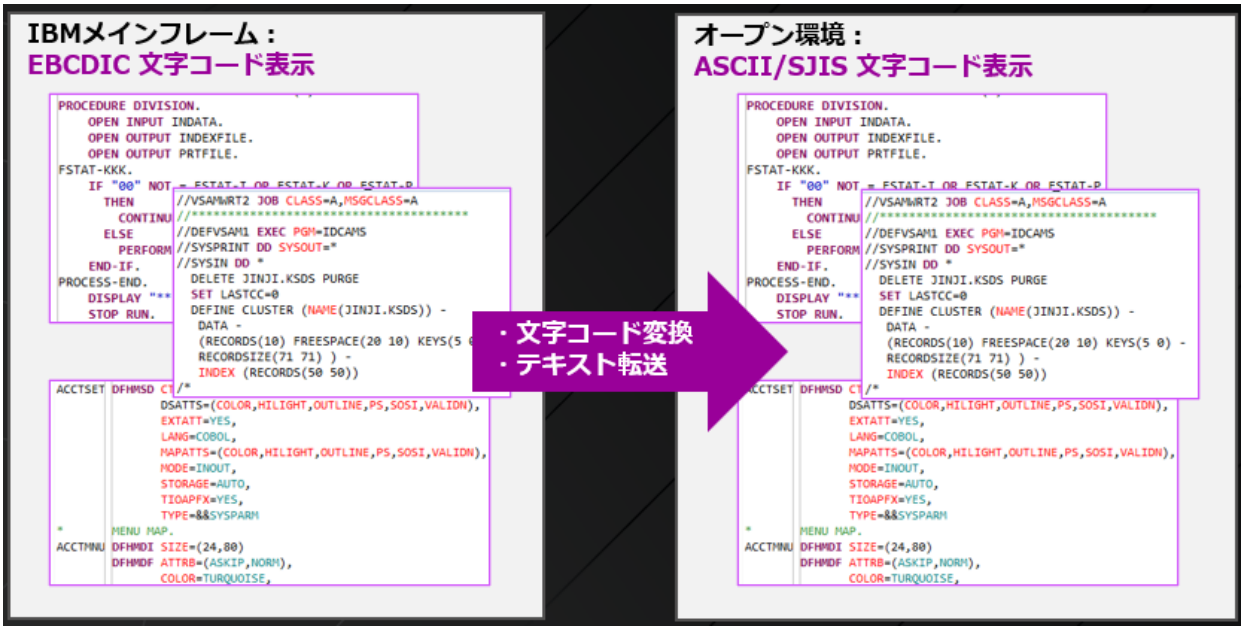
VS COBOL II および COBOL/370 の規則に従って切り捨てを行います。

8. 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 IBM メインフレームからオープン環境へ転送します。

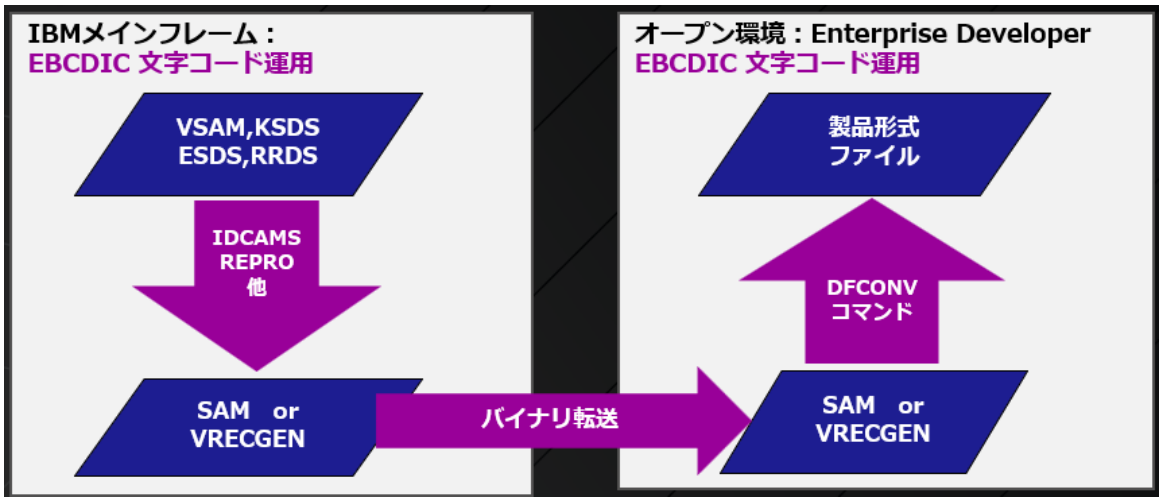
1)アプリケーション

IBM メインフレームでは COBOL ソース、コピー句、JCL、画面定義などのソース類に EBCDIC 文字コードを使用していますが、オープン環境の OS では ASCII/SJIS 文字コードを使用するため、ASCII/SJIS 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

EBCDIC 文字コードデータを使用することを前提にテストを計画している場合は、IBM メインフレームで 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。転送後、VRECGEN 形式ファイルは製品に含まれているコマンドを利用して、製品形式のファイルへ変換します。

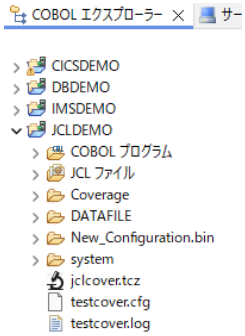


9. COBOL ソースのコンパイル

コンパイルは、統合開発環境である Eclipse、Visual Studio、Visual Studio Code やコマンドラインから実行可能です。本書では Eclipse を使用して、32 ビットアプリケーション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

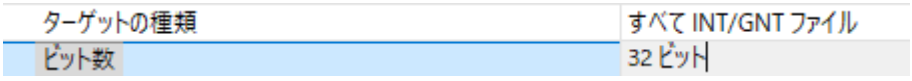
メニューから Enterprise Developer for Eclipse を起動後、メインフレーム COBOL プロジェクトを新規作成して ASCII/SJIS 文字コードに変換されたソース類をインポートします。



2)コンパイラ指令の設定

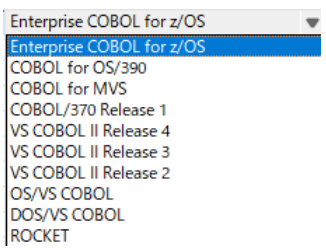
プロジェクトのプロパティでコンパイラ指令を指定します。

ターゲットに動的ロード形式の実行モジュールとして GNT を指定し、32 ビットアプリケーションを指定します。



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、文字セットには EBCDIC を指定します。IBM メインフレームのコンパイラバージョンを COBOL 方言に指定します。この指定により互換性を持つコンパイラ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を指定します。



文字セット	EBCDIC
ソース エンコーディング	ANSI
COBOL 方言	Enterprise COBOL for z/OS

3)コンパイルの実行

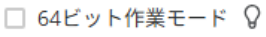
プロジェクトをコンパイルして実行モジュールを生成します。

10. 開発用実行環境の設定

Enterprise Developer に搭載されている開発用実行環境である Enterprise Server インスタンスを、32 ビット稼働として作成し、コンパイルにより生成された 32 ビットの実行モジュールを実行します。

ア...	名前	タイプ	ステータス	64ビット	MSS有効	セキュリティ
	JCL32	Region	Stopped		✓	デフォルト

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の MF_CHARSET=E を指定します。これにより、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。データとして扱わない、例えば JES 機能のシスログなどは ASCII/SJIS 文字コードで出力されます。

構成情報

```
[ES-Environment]
proj=C:\work\JCLDEMO
MFACCCGI_CHARSET=Shift_JIS
MF_CHARSET=E
```

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

11. JCL の実行と結果確認

EBCDIC モードで稼働する実行モジュールと 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

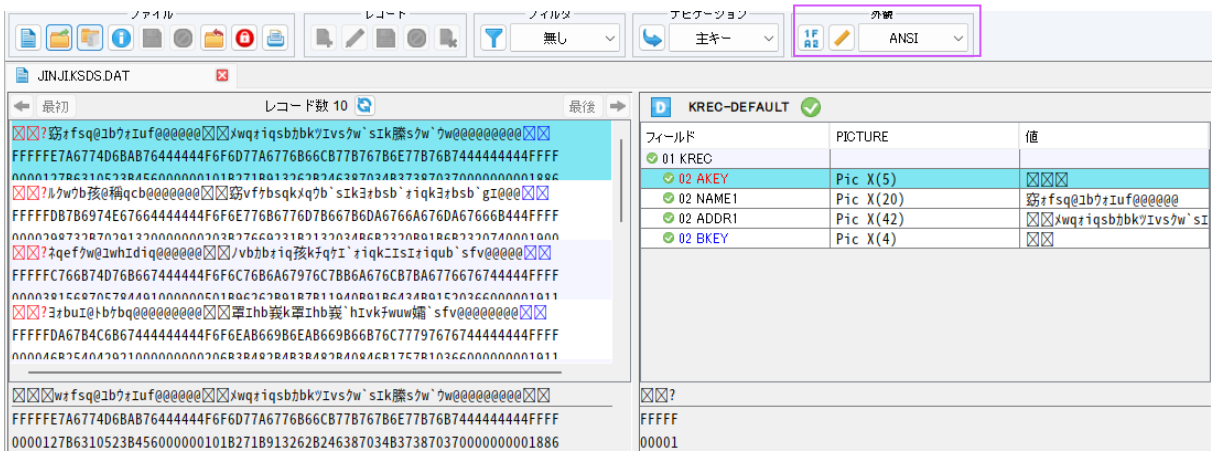
下記のサンプル JCL を実行します。カタログされている VSAM ファイルの JINJI.KSDS を削除後、再生成し、JCL に書かれたデータを SORT 後、JINJI.KSDS ファイルへ書き込む処理を行います。

```
//VSAMWRT2 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 *
00009Shiki Masaoka       5-5,Dogo Onsen,Matsuyama-shi,Ehime-ken     1870
00001Soseki Natsume      1-1,Koishikawa,Bunkyo-ku,Tokyo-to          1886
00007Ogai Mori           3-1,Rintaro-cho,Tsuwano-shi,Shimane-ken    1886
00002Ryotaro Shiba       2-3,Sonezaki,Kita-ku,Osaka-shi,Osaka-fu    1900
00006Jirocho Shimizu     6-6,Jiro-cho,Shimizu-shi,Shizuoka-ken      1800
00005Eiji Yoshikawa      9-3,Miyamotomura,Mimasaka-gun,Okayama-ken 1920
00004Osamu Dazai         2-6,Tsugaru,Tsugaru-gun,Aomori-ken         1911
00008Ryoma Sakamoto      1-1,Harimayabashi,Kochi-shi,Kochi-ken      1820
00010Yukichi Fukuzawa    8-8,Keio-cho,Nakatsu-shi,Oita-ken          1835
00003Hideyo Noguchi      5-1,Inawashiro,Aizu-shi,Fukushima-ken      1911
/*
//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)
/*
```

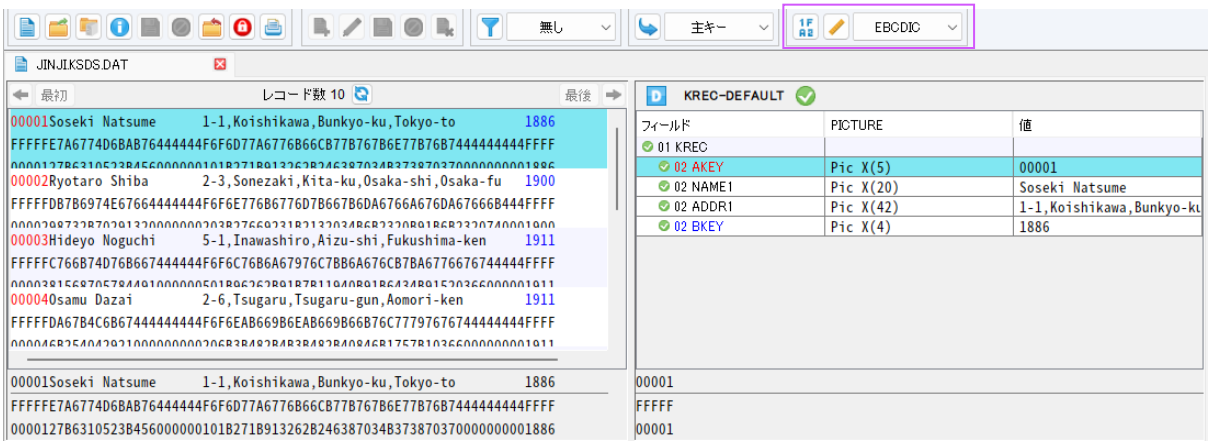
2）入力データの内容

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

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



EBCDIC 表示へ切り替えて内容を表示すると可視化され、HEX 表示でも EBCDIC 文字コードが格納されていることが確認できます。



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 出力の結果は

```
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を、共通管理画面から確認します。カタログされたファイル内容を表示するとコンバート機能が動作し、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はできません。

【共通管理画面から表示したデータ】

【テキストエディターから表示したデータ】

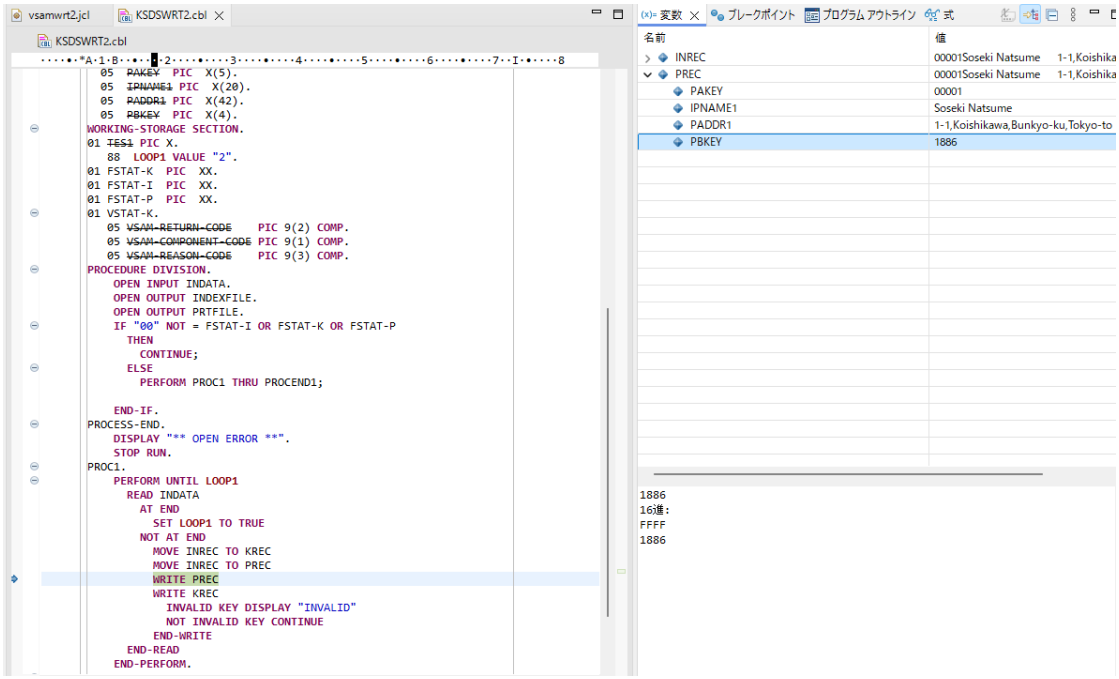
シスログの内容は ASCII/SJIS 文字コードで出力されるため、テキストエディターへ表示しても読むことができますが、SYSPRINT、SYSOUT はデータとして扱われ、EBCDIC 文字コードで出力されます。

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がある場合でも修正の必要はありません。また、インデックスや可変長データファイルに関しては製品形式に変換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が確認できました。

12. 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 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専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 文字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/SJIS 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即時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。

13. 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 EBCDIC 文字コード以外であれば、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データベースの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 Enterprise Developer コマンドプロンプトからコンパイルします。

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

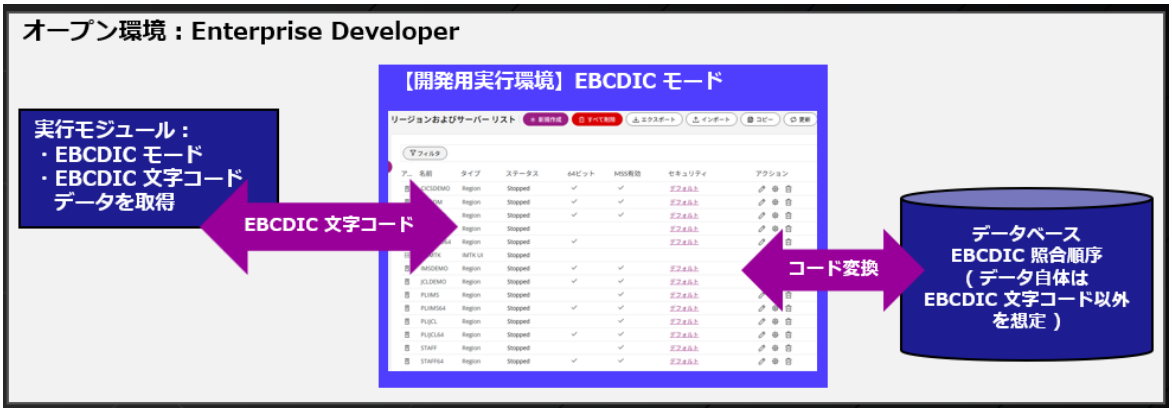
コマンド）ebcdicdb;

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

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 文字コードになります。



14. IBM メインフレームと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 IBM メインフレームでコンパイルして結果を比較します。

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 のコンパイル結果

レベル E のコンパイルエラーが発生しており、実行ファイルは生成されています。

```
[cobol] * チェック終了 - 重大なエラーはありません - コード生成を開始します
[cobol]     54      MOVE "AAAAA" TO SAMP-9.
[cobol] COBCH1026E 送り出し側が文字定数である - 代わりにゼロを転記する
[cobol] Compilation complete with 0 errors, 1 warnings, 0 nc
```

3)IBM 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差異が発生します。これにより、コンパイルが中断されることなく、IBM メインフレーム資源を使用する前に多くのエラーを検出し、修正することが可能になります。

15. 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、IBM 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、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、開発用実行環境を使用すれば、個々のユーザが“疑似 IBM 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。

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、Visual Studio、Visual Studio Code を使用することにより、世代を超えた技術の継承が現実的になります。

16. 補足:稼働環境

本書は下記環境で実施されました。

1)OS

Windows 11 Pro

2)プロセッサ

13th Gen Intel(R) Core(TM) i7-13800H (2.92 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)製品バージョン

Enterprise Developer 11.0J

注意)Enterprise Server 11.0J と同等の開発用実行環境を含んでいます。

5)製品マニュアルバージョン

Enterprise Developer 11.0J for Eclipse