Rocket® software

Enterprise Developer を使用した PL/I 開発業務のモダナイゼーション

Version.2
Last updated: 2026 年4月

開発業務のモダナイゼーション

Rocket software

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 メインフレームで稼働する基幹システムは、ビジネスに欠くことのできない重要な経営資産として現在も世界のビジネストランザクションの主要な業務を担い、日々稼働し続けています。しかし、開発業務には IBM メインフレームの多くの資源が必要となり、これによるコストの増加や、古い開発スタイルにより、作業の効率化が図りづらいなどの問題があることも事実です。例えば、クラウドなどのオープン環境へ基幹システムを移行した場合、開発スタイルはどのように変化するのでしょうか。

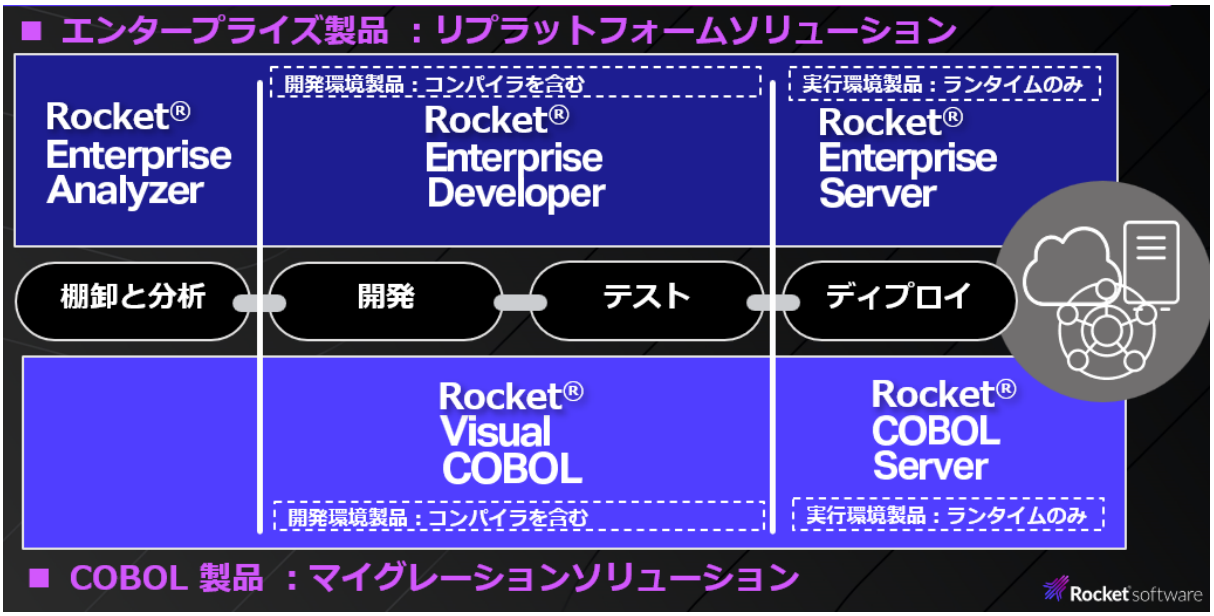
本書では、弊社のモダナイゼーション支援製品である Enterprise Developer を使用した、リプラットフォーム後の PL/I アプリケーションの開発スタイルについて解説します。

2. モダナイゼーション支援製品の分類

弊社のモダナイゼーション支援製品は、過去の投資を無駄にせず、実績のある PL/I アプリケーションをオープン環境で再活用できる開発/実行環境製品です。

製品は2つに分類されており、エンタープライズ製品群には、静的解析ツールの Enterprise Analyzer、コンパイラと開発用実行環境を含む開発環境製品の Enterprise Developer、ランタイムのみを含む実行環境製品の Enterprise Server が含まれ、COBOL と PL/I アプリケーション、IBM メインフレームのミドルウェア互換機能をサポートするリプラットフォームを支援します。

COBOL 製品群は、コンパイラと開発用実行環境を含む開発環境製品の Visual COBOL と、ランタイムのみを含む実行環境製品の COBOL Server があり、IBM、国産メインフレーム、オープンレガシーからの COBOL アプリケーションの移行を支援します。

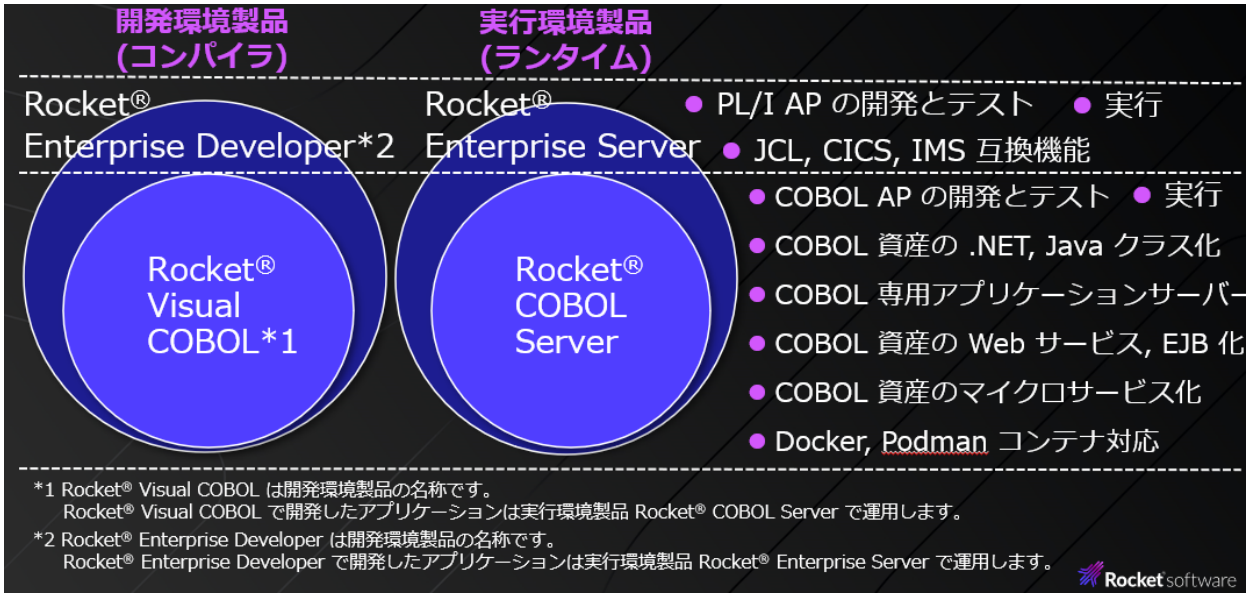


3. モダナイゼーション支援製品の関係性

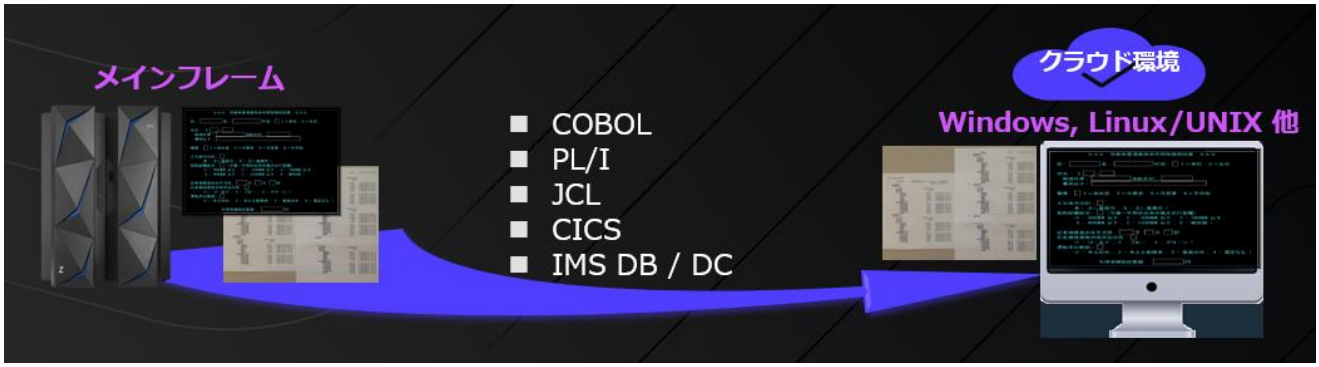
エンタープライズ製品は COBOL 製品の全機能を含む上位製品となり、リプラットフォーム後も新しい技術を取り入れながら、更なるモダナイゼーションを目指すことができます。

4. モダナイゼーション支援製品の機能分布

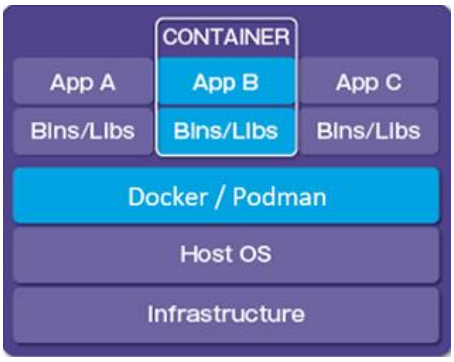
既存の資産を将来に渡り有効利用できるように、COBOL や PL/I アプリケーションを新しい技術と連携させる機能が製品には含まれています。



エンタープライズ製品は、PL/I 言語のサポートや IBM メインフレームの JES、CICS、IMS 互換性機能を持ち、オープン環境へ移行後も JCL、EXEC CICS、EXEC DLI、CBLTDLI、PLITDLI 構文などを利用することができ、必要最低限のコストと期間でアプリケーションの品質を損なうことなく、IBM メインフレームからのリプラットフォームを実現することができます。

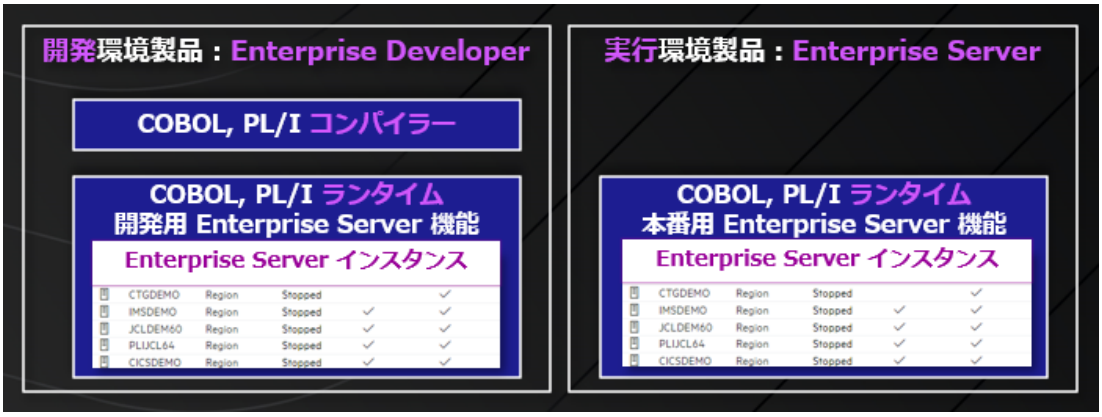


また、クラウド上でマイクロサービスを導入する際に欠かせない
コンテナ型仮想化イメージもご提供しており、
PL/I アプリケーションをコンテナで稼働させることができます。



5. 製品の利用用途

COBOL、PL/I ランタイムのみを含む実行環境製品の Enterprise Server は、コンパイルを必要としない本番環境に使用する製品です。アプリケーションを実行させる単位である Enterprise Server インスタンスは、例えば JCL を対象としたバッチ用、IMS、CICS などのオンライン用など、運用用途に合わせて複数設定することができます。



開発環境製品の Enterprise Developer は Enterprise Server と同等の機能を持つ開発用実行環境を含んでおり、IBM メインフレームのミドルウェア互換機能を利用した単体テストを実行することができます。

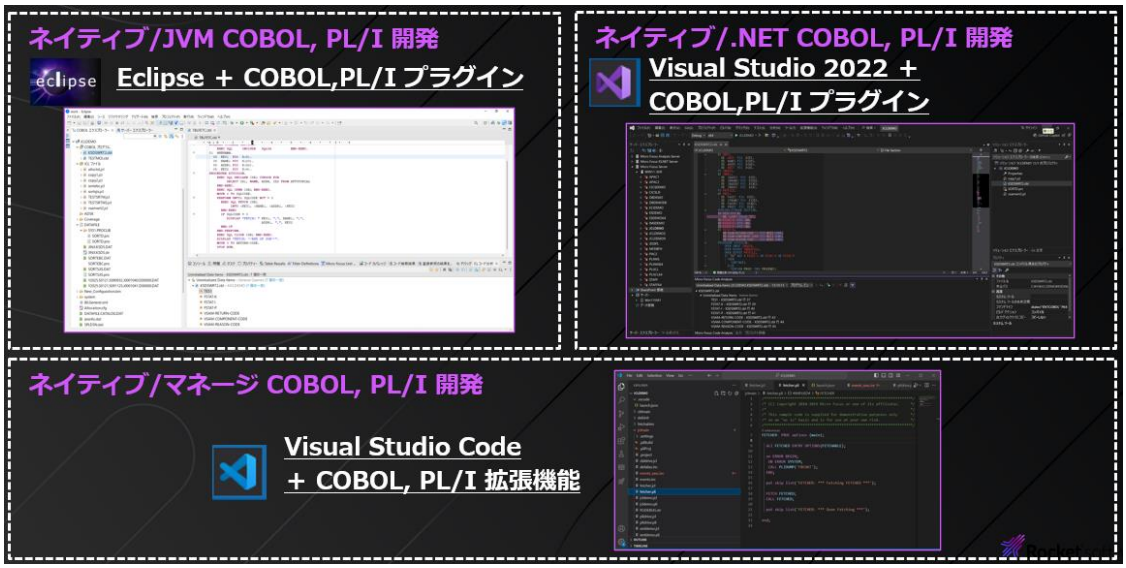
6. Enterprise Developer を利用した開発業務

PL/I アプリケーションをオープン環境へ移行後は、開発環境製品である Enterprise Developer を使用して開発業務を行います。Enterprise Developer が持つ様々な機能を利用することにより、作業の効率化が計れ、開発用実行環境を利用することにより、IBM メインフレームを独り占めするイメージで単体テストを実施することができます。

この章では、Enterprise Developer の具体的な開発機能についてご紹介します。COBOL の開発機能については、「COBOL 開発業務のモダナイゼーション」ホワイトペーパーをご参照ください。また、リプラットフォームにおける一般的な手順や注意点については「COBOL、PL/I アプリケーションのリプラットフォーム手順と注意点」ホワイトペーパーをご参照ください。

6.1 統合開発環境（IDE）

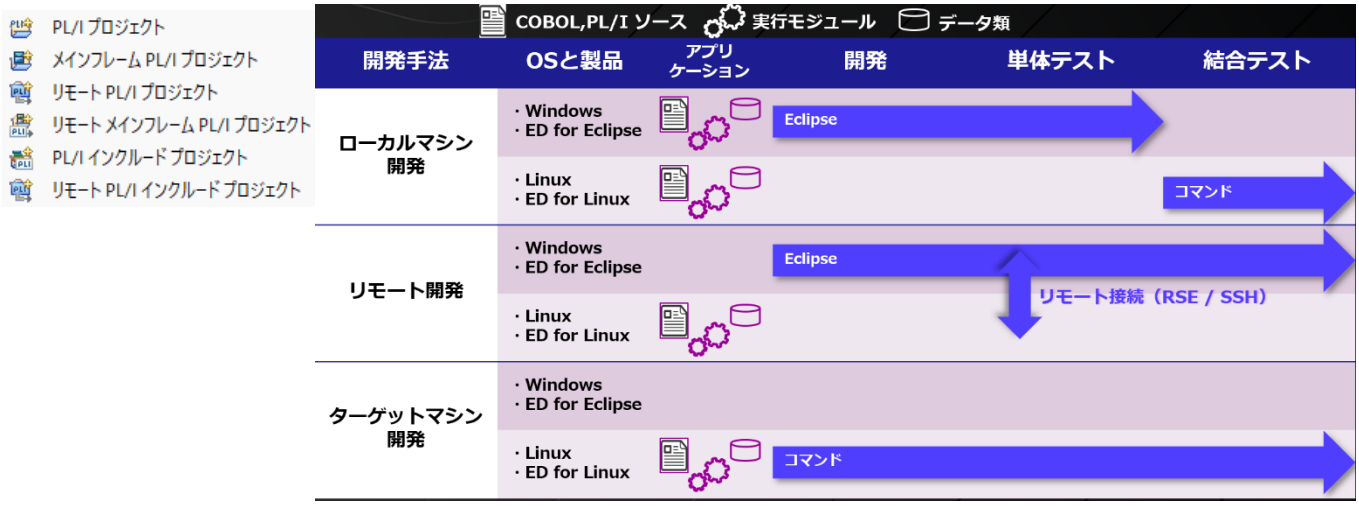
業界標準の IDE である Eclipse、Visual Studio、Visual Studio Code を利用して、コーディング、コンパイル、デバッグ、テストを効率的に行うことができます。また、テキストエディターでソースを編集後、コマンドを利用したコンパイルもでき、開発者に合わせた開発スタイルを選択できます。



IDE ではデバッガを利用することにより、ステップ実行しながら変数の値を確認するなど、開発業務の効率化や品質を担保することができます。

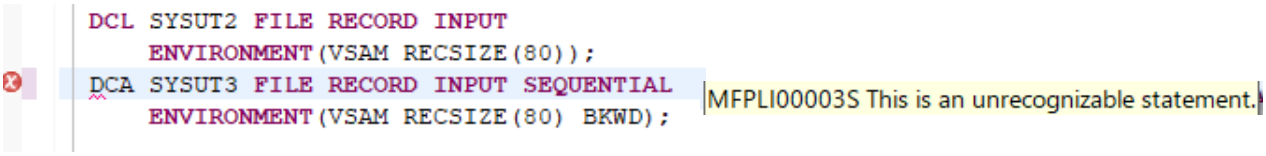
6.2 PL/I プロジェクト

Eclipse と Visual Studio では PL/I 専用のプロジェクトを作成して、関連する PL/I ソースや JCL などを配置します。また、Eclipse から Linux マシンに接続し、Linux マシンに配備したソースを編集後、実行モジュールを Linux マシンに生成できるリモートプロジェクトも利用できます。



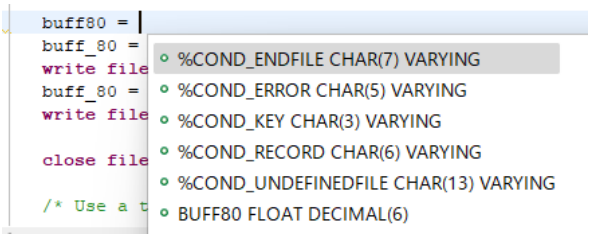
6.3 PL/I エディタ機能

PL/I プロジェクトに配置されたソース類は、PL/I 専用エディタを使用してコーディングを行います。これにより、コーディング時に即時エラー判定が可能なバックグランドパーシング機能を利用することができます。

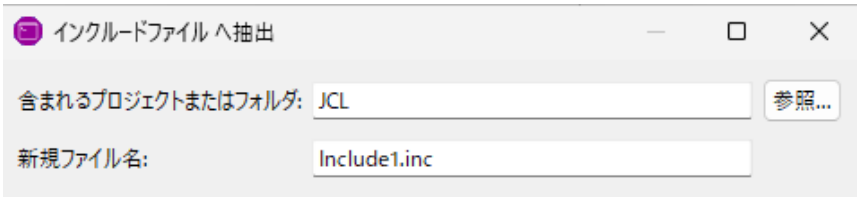


また、項目定義の位置へジャンプすることや、マウスオーバーによるデータ属性の確認、転送先候補の表示ができ、コーディングミスを未然に防ぐことで、開発工数の削減に貢献します。また、ソースコードのリファクタリングも可能です。

転送先候補を提供するコンテンツアシスト例）



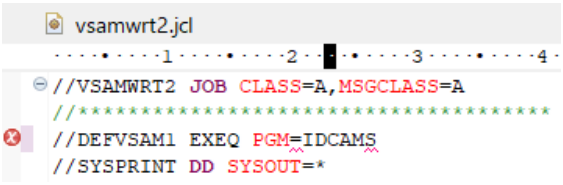
ソースのリファクタリング例）



参照箇所の検索例）

6.4 JCL エディタ機能

JCL 専用のエディタを利用してコーディングを行うことができます。PL/I ソースと同様に、即時エラー判定が可能なバックグランドパーシング機能を利用することができます。



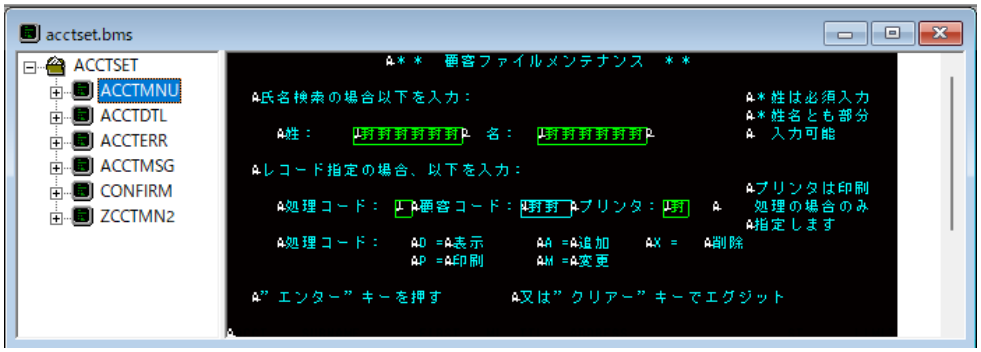
6.5 CICS 構文のサポート

EXEC CICS 構文や API、SPI コマンドの大部分をサポートしています。BMS 画面定義もサポートしており、画面イメージでメンテナンス可能な BMS ペインターや、Eclipse の BMS プレビューを利用できます。

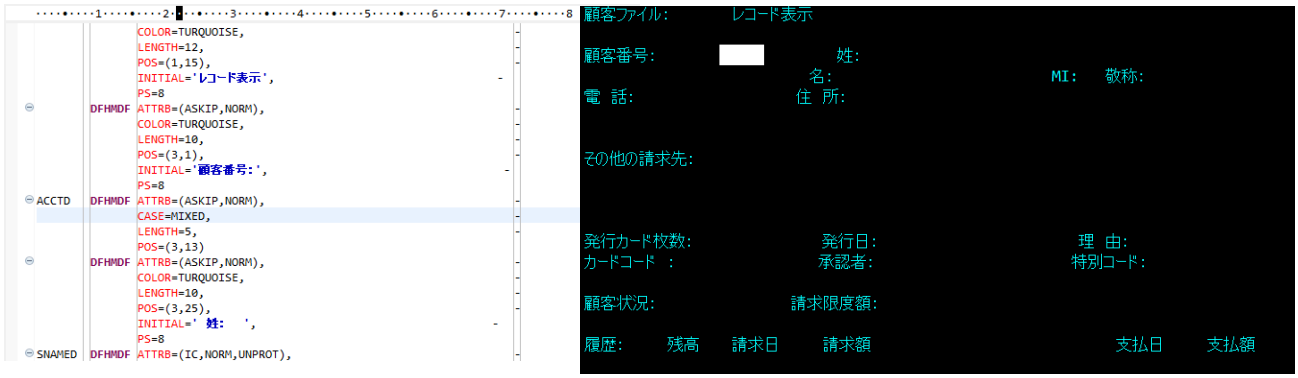
COBOL ソースコード例）

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

BMS ペインター例）



Eclipse の BMS プレビュー例）



6.6 IMS 構文のサポート

CALL インターフェイスである PLITDLI や EXEC DLI といった IMS プログラム構文や MFS 定義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。また、IBM メインフレームで使用している IMS 資源定義マクロをそのまま利用することができるため、新しくマクロを作成する必要がありません。JCL で使用する MPP、BMP、DLI もサポートしています。

PL/I ソースコード例）

```
ADDRBOOK: PROC(P_PCB_LT,P_ALT_PCB,P_DEMO_PCB) OPTIONS(MAIN);
DCL P_PCB_LT   POINTER;
DCL P_ALT_PCB  POINTER;
DCL P_DEMO_PCB POINTER;

/* Set starting position */
CALL PLITDLI( FOUR, GU, demo_pcb, NAME_AND_ADDRESS, SSA_SET_START);
```

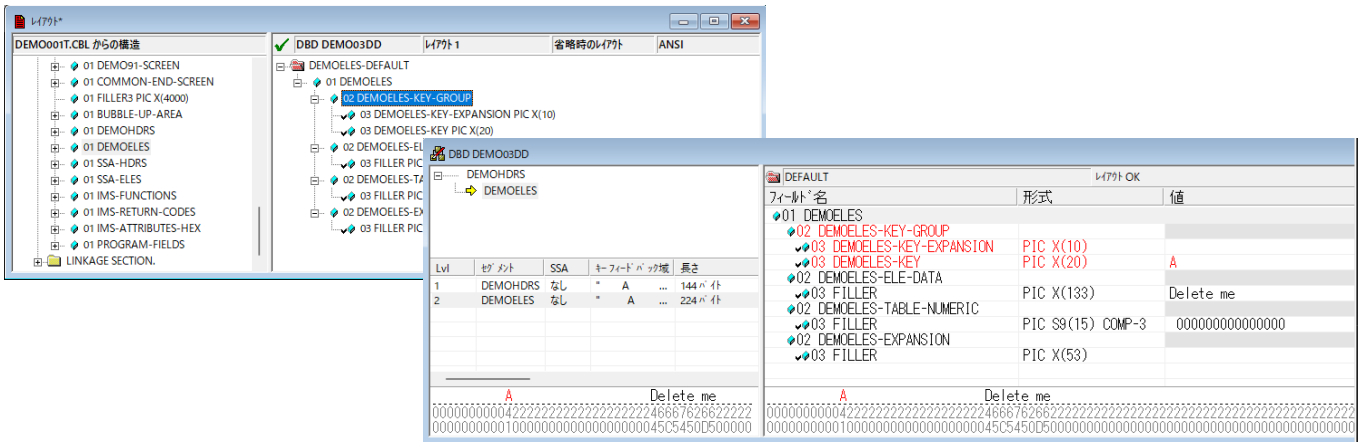
MFS 定義例）

```
OTDEMO91 MSG   TYPE=OUTPUT,SOR=(DEMO91,IGNORE),FILL=NULL,PAGE=YES,
               NXT=INDEMO91
         SEG
         MFLD  FLD0000,LTH=0079
         MFLD  FLD0001,LTH=0008
         MFLD  FLD0002,LTH=0035
         MFLD  LTERM,LTH=0008
```

JCL 例）

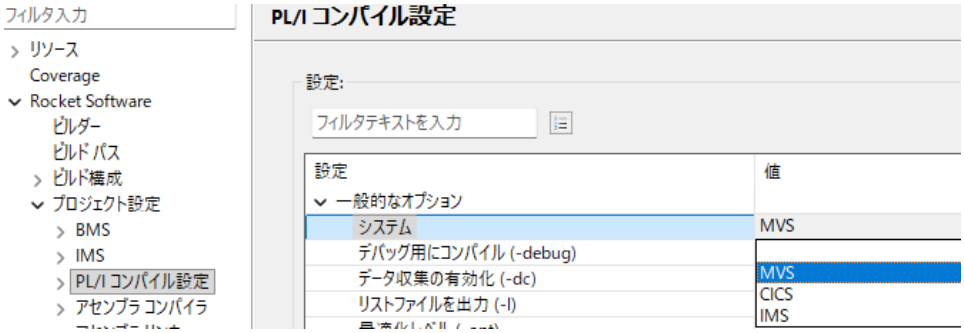
```
//S01       EXEC PGM=DFSRRC00,REGION=4M,
//            PARM='BMP,DEMO001B,DEMO001T,,,,,,,,CDLI,,N,N'
```

また、IMS データのセグメントレイアウトに沿ってデータのメンテナンスを行うことができるデータファイルツールも付属されており、レコードレイアウトを利用してデータを管理することができます。



6.7 PL/I コンパイラ指令

PL/I プロジェクトまたはソースファイルにプロパティとしてコンパイラ指令を指定します。どのミドルウェア制御の下で実行されるのか MVS、CICS、IMS から選択することで、メインフレームとの互換性を保つことができます。可視化されたプロパティ指定やその説明により、誰もが容易に指定内容を理解することができます。



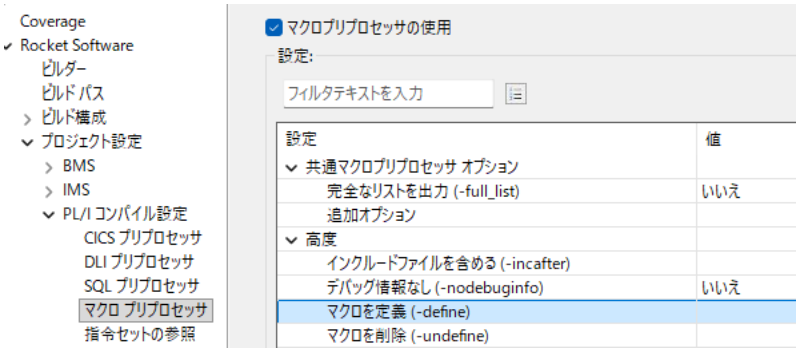
注意）

コマンドによるコンパイルとリンクや注意点については後述の「PL/I コンパイルとリンクにおける注意点」の章で詳しくご説明します。

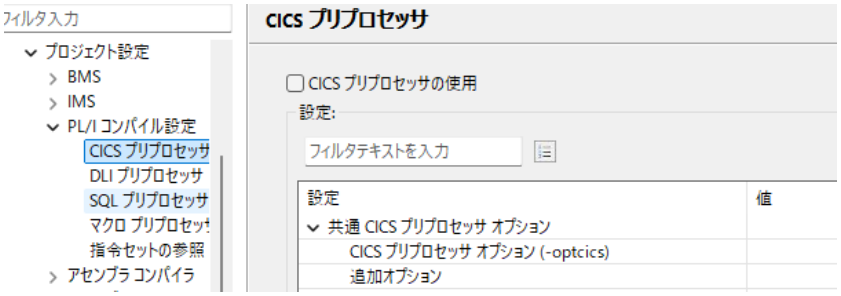
6.8 PL/I マクロ プリプロセッサ機能

PL/I マクロ プリプロセッサを使用することができます。この機能を利用すると、コンパイルの前にマクロ プリプロセッサが実行され、その出力を PL/I コンパイラに引き渡すことができます。独自のプリプロセッサや EXEC CICS、IMS で使用する EXEC DLI や EXEC SQL プリプロセッサも利用することができます。

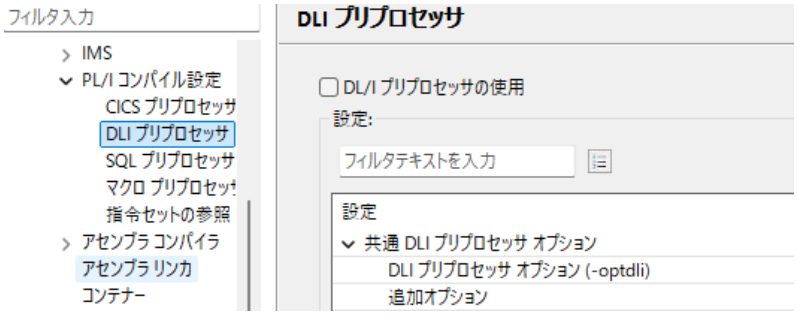
マクロ プリプロセッサ指定画面例）



CICS プリプロセッサ指定画面例）



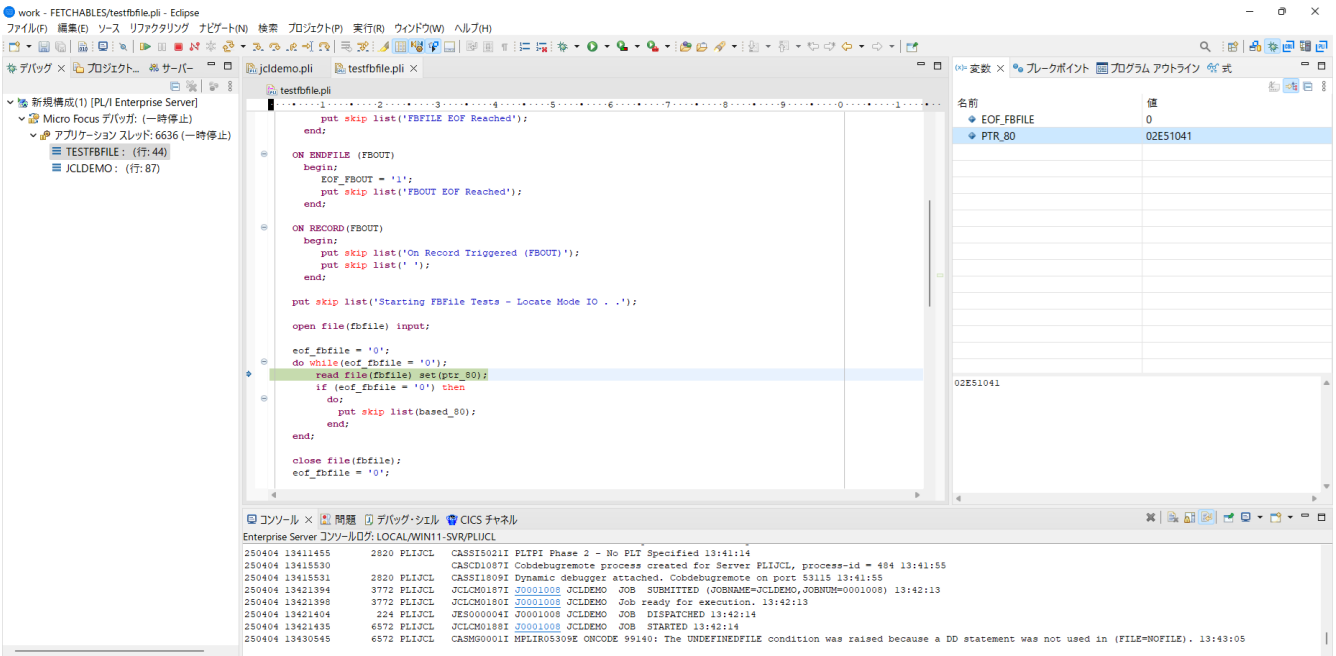
DLI プリプロセッサ指定画面例）



6.9 デバッグ機能

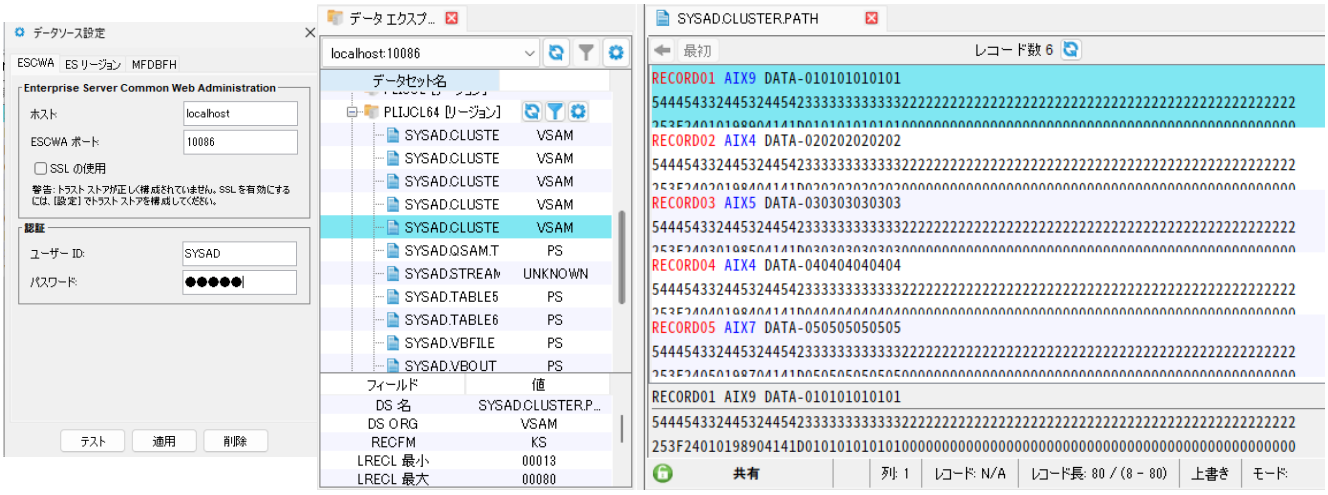
IDE 内部のデバッガを利用して、PL/I ソースをステップ実行、デバッグができます。変数の値を 16 進数で確認することや、動的な通過ルートの確認ができ、単体テストの品質向上に貢献できる機能です。

Eclipse のデバッグ画面例）



6.10 データファイルツール

EBCDIC 文字コードと ASCII/SJIS 文字コートを持つデータをメンテナンスできるツールです。プログラムから選択したレコードレイアウトを適用したデータメンテナンスや、16 進数でのメンテナンスも可能です。また、Enterprise Server インスタンスの JES カタログファイルと連携したデータメンテナンスを行うことができます。



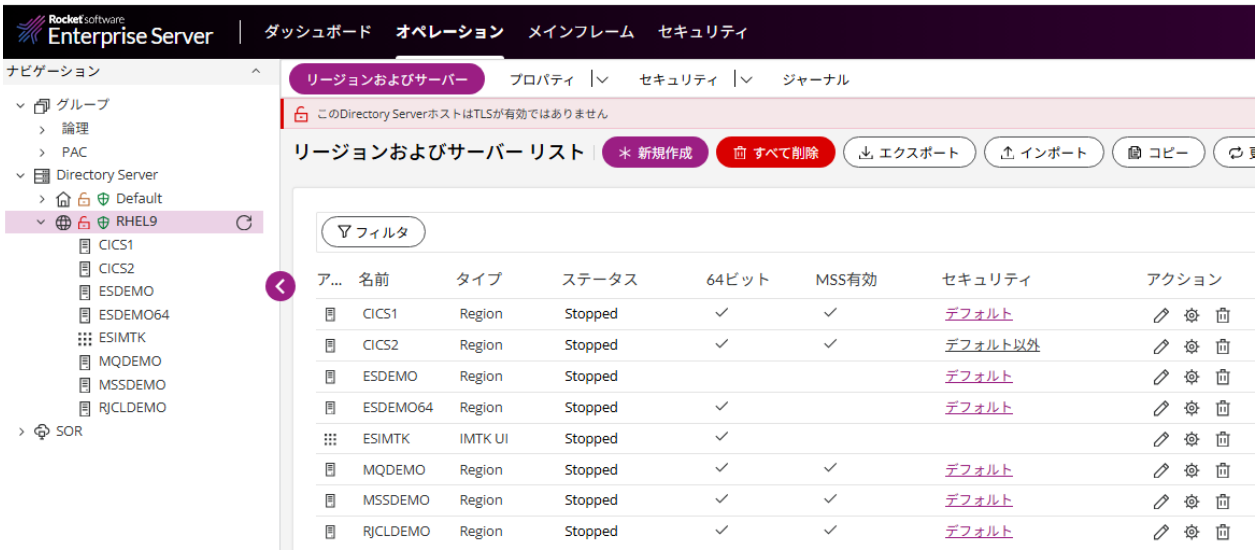
7. Enterprise Server を利用した実行

Enterprise Server には利便性を追求した機能や、IBM メインフレームとの互換性を保つための機能、運用と管理をサポートする機能など、様々な機能が備わっています。この章では実行環境の機能についてご説明します。

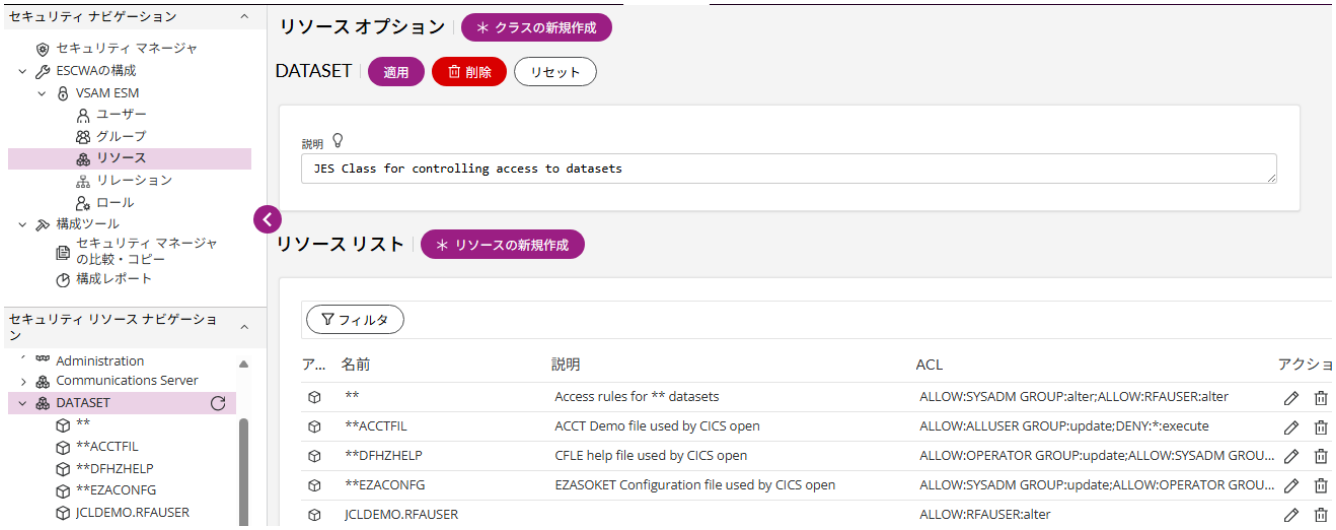
7.1 Enterprise Server Common Web Administration（略称 ESCWA）

実行結果など、運用管理者が参照する Enterprise Server インスタンスの管理画面です。Web ベースの画面を利用することにより、異なる筐体、かつ異なる OS を持つホストマシンに作成した Enterprise Server インスタンスを一括管理することができます。

Windows と Linux マシンの Enterprise Server インスタンス管理例）



また、製品に付随しているデフォルトの VSAM ESM セキュリティを使用すれば、ユーザー、グループ単位での細やかなアクセス制限を実施できます。



7.2 JES 機能

個々の Enterprise Server インスタンスが持つ JES 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 ユーティリティの多く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。

JES スプール画面例）

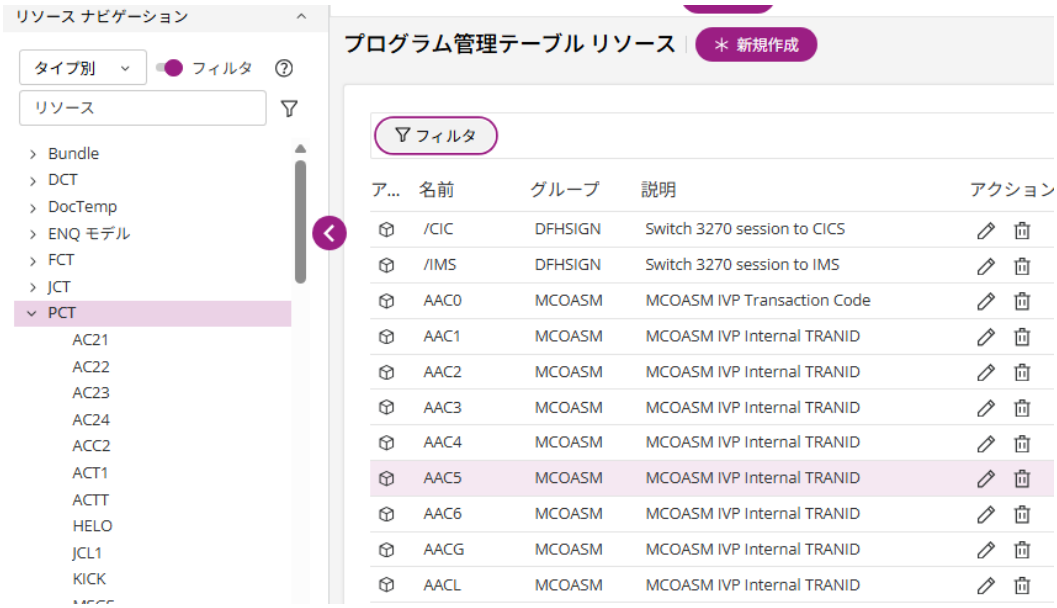
JES カタログファイル DCB 情報例）

JES カタログプロシージャ情報例）

7.3 CICS 機能

個々の Enterprise Server インスタンスが TP モニターの役割を持つため、OLTP ツールを別途用意する必要はありません。TN3270 エミュレータを 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 ファイルに定義された画面が表示されます。また、CICS リソース定義ファイルに登録された PCT や FCT などを使用したアプリケーションの実行が可能です。

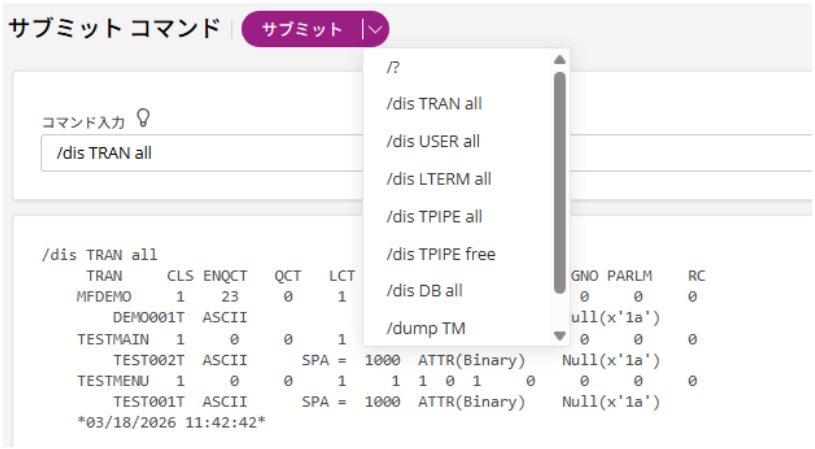
CICS リソース管理画面例）



7.4 IMS 機能

IMS オンライン処理においても、CICS 機能と同様に Enterprise Server インスタンスが TP モニターの役割を持つため、OLTP ツールを別途用意する必要はありません。TN3270 エミュレータを 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 MFS ファイルに定義された画面が表示され、IMS トランザクションを実行できます。また、IMS データベース制御コマンドを利用することもできます。

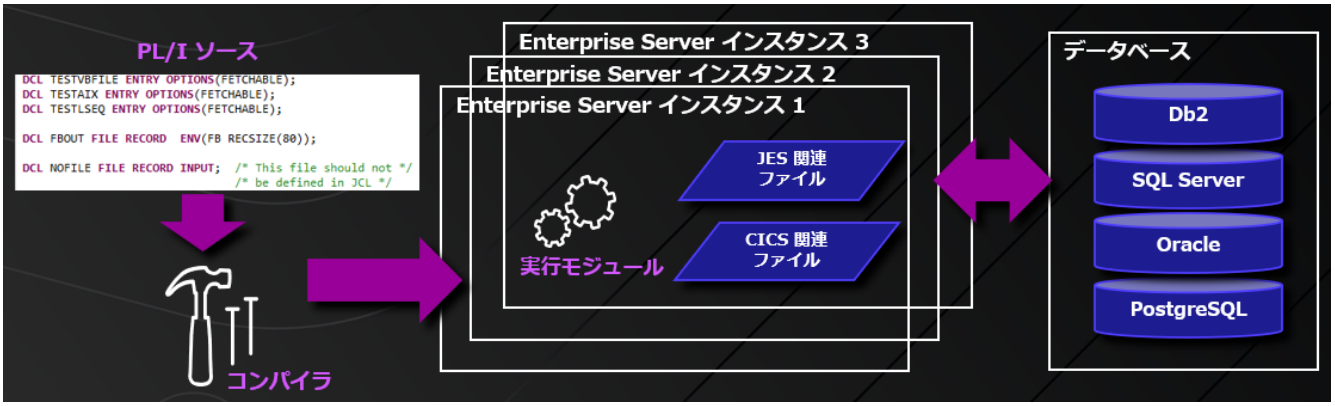
IMS コントロール画面例）



IMS トランザクション例）

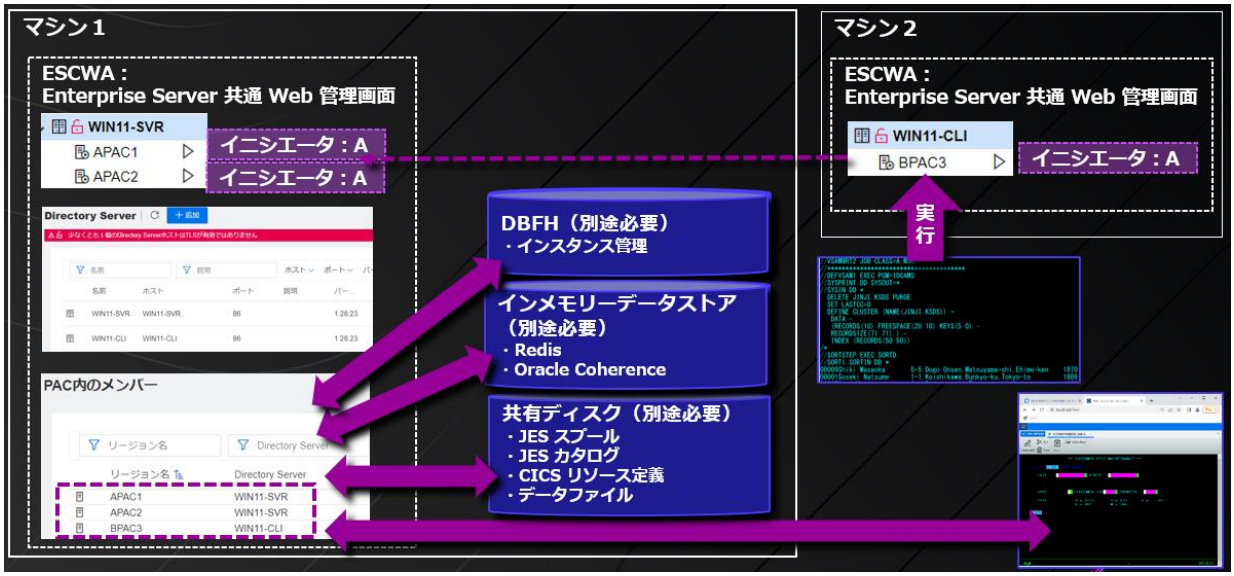
7.5 Database File Handler(DBFH)機能

ソース記述の変更なく、JCL、CICS で使用するデータファイルをデータベースに格納できる機能です。これを利用することで、JES 機能で使用するカタログファイルや CICS の FCT に登録されたファイルなどを、複数の Enterprise Server インスタンス間で共有することができます。ただし、データの格納形式は一般のテーブル項目属性とは異なります。



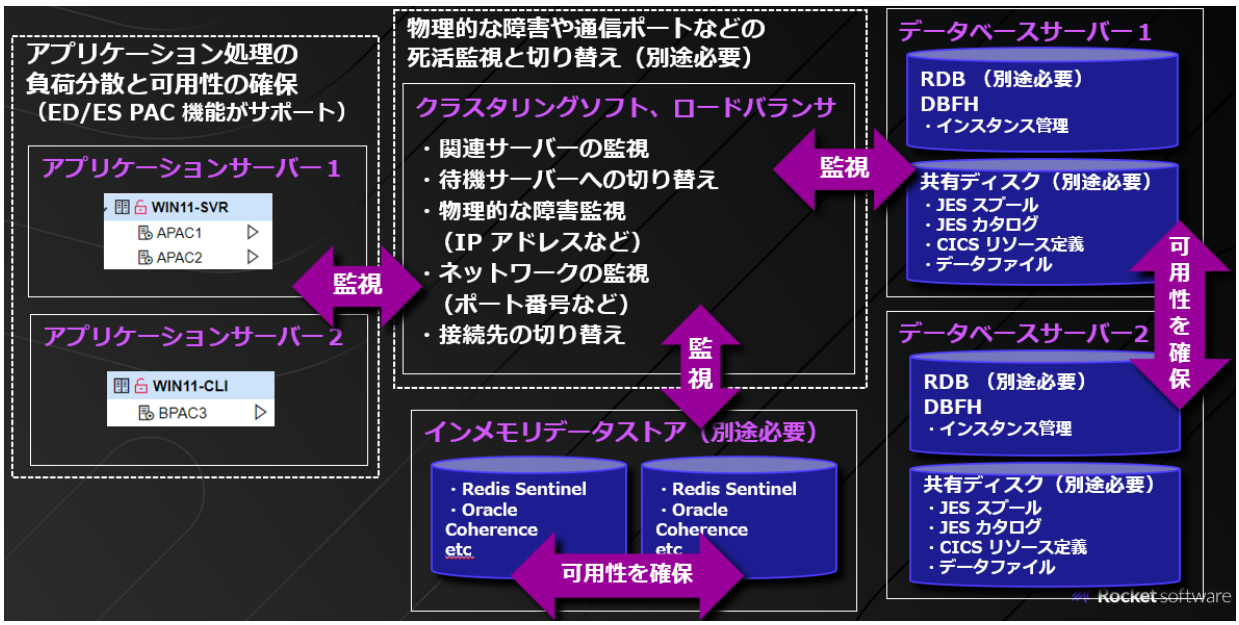
7.6 スケールアウト パフォーマンス/可用性クラスター機能(PAC)機能

複数の Enterprise Server インスタンスをグループ化し、JCL の負荷分散を実現することができます。OS、製品バージョンなどの前提条件が一致すれば、異なるホストマシンに存在する Enterprise Server インスタンスをグループに含めることができ、物理的なリスクも回避した運用を実現することができます。また、連携をとるために、RDB とインメモリーデータストアを使用します。



ただし、製品機能のみでシステム全体の可用性を確保できるわけではありません。RDB やインメモリーデータストアの可用性、使用ポートの死活監視などの周辺ツールが必要になります。

システム構成の例）



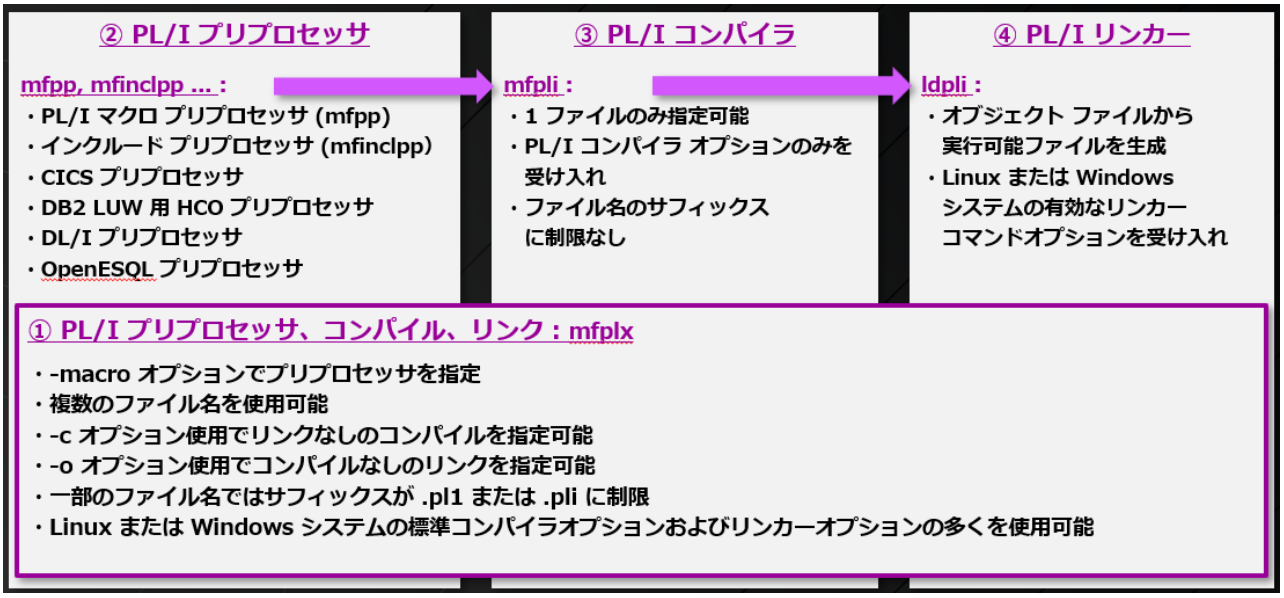
8. PL/I コンパイルとリンクにおける注意点

Enterprise Developer を利用することにより、リプラットフォームを目的とした PL/I アプリケーションのオープン化を実現することができますが、Enterprise Developer がサポートする PL/I は ANSI 1987 規格に準拠した Open PL/I が起源となっており、一部サポートがないものについては独自のプリコンパイラを使用する、もしくはプログラムを修正するなどの対策が必要となります。この章では、よくあるご質問をもとに、その注意点やコンパイルとリンクの方法などをご説明します。製品のバージョンアップに伴い、利用可能な構文や機能が追加されますので、サポート内容については必ず利用バージョンに合った製品マニュアルをご確認ください。

8.1 コンパイルとリンクコマンド

IDE を利用せずに makefile やスクリプトによるコンパイルやリンクを行う場合は、コマンドを使用することになります。Enterprise Developer はプリプロセッサによるマクロや、インクルード文の展開、コンパイラによるオブジェクトファイルの生成、リンカーによる実行モジュールの生成など、様々なコマンドとそのオプションを提供しています。

コマンドの概要）



① プリプロセッサからリンクまで実行可能なコマンド：mfplx

PL/I ソースファイルから、プリプロセッサ、コンパイラ、リンカーを経由して実行モジュールを生成するコマンドです。複数のファイル名を指定でき、mfpli コンパイラオプションのほか、-c および-o などの、Linux または Windows システムの標準コンパイラオプションおよびリンカーオプションの多くを使用できます。コンパイルのみ、リンクのみの実行も可能です。ただし、一部のファイル名ではサフィックスが.pl1 または.pli に制限されます。

コマンド例 1）

mfplx -db2 -cics sample.pli sample2.pli -#

コマンド例 2）マクロ展開後のソースを保持

mfplx -deb -macro -list -verbose sample.pli -incl -isuffix .inc -nodebuginfo -pp sample.pp

例えば、コンパイルだけを行いたい場合は、②以降の各フェーズに対応するコマンドを単体で実行することもできます。

② プリプロセッサコマンド：mfpp, mfinclpp 他

マクロなどが展開されたソースファイル（.pp）を生成するコマンドです。このコマンドを使用することもできますが、mfplx コマンドを利用した-macro オプションの指定によるプリプロセッサの実行を推奨しています。

コマンド例）

mfpp sample.pli -pp sample.pp

③ コンパイルコマンド：mfpli

オブジェクトファイル（.obj）を生成するコマンドです。指定できるファイル名は 1 つだけです。サフィックスに制限はなく、PL/I コンパイラオプションのみ指定可能です。

コマンド例）

mfpli sample.pp -o sample.obj

④ リンクコマンド：ldpli

オブジェクトファイルから実行モジュールを生成するコマンドです。Linux または Windows システムの有効なリンカーコマンドオプションはすべて受け入れられます。

コマンド例）

ldpli -db2 sample.obj sample2.obj

IDE を利用すれば、コマンドを意識せず、プロジェクトプロパティ指定に沿った実行モジュールを生成することができます。

8.2 PL/I コーディングの注意点

メインフレームで稼動している PL/I コーディングの解釈と異なる代表的な非互換点を説明します。次に挙げるコーディングが存在する場合は実際に影響出るか否かなど、確認と対策が必要になります。製品バージョンに合わせたマニュアルトップから“リファレンス＞メインフレーム リファレンス＞Open PL/I リファレンス＞Open PL/I 言語リファレンス マニュアル＞Open PL/I でサポートされていない機能”箇所をご参照ください。

① 静的リンクと動的ロード

Enterprise Developer の PL/I コンパイラは、ENTRY 属性のコーディング方法により CALL ステートメントの呼出しを静的リンクもしくは動的ロードと認識し、実行モジュールを生成します。

静的リンクの呼出しとしてコンパイルするコーディング）

```
DCL XXXXXX ENTRY;
```

呼出し先が存在しない場合のエラー発生タイミング：リンク時

動的ロードの呼出しとしてコンパイルするコーディング）

```
DCL XXXXXX ENTRY OPTIONS(FETCHABLE);
```

呼出し先が存在しない場合のエラー発生タイミング：実行時

② 非標準文字の使用

本来、PL/I 言語は識別名に@を許容していますが、Windows の LINK.exe や Linux の ld コマンドの仕様に由来する外部シンボルの制限から、Windows の DLL または Linux の.so にリンクして動的 CALL を行う場合にのみ@を外部シンボルとして使用することができません。また、AIX ではリンカーが$記号とアセンブラを混同するため、正常にリンクできない可能性があります。

③ 配列のクロスセクション

配列のクロスセクションは、代入文、PUT/GET 文、および組み込み関数でサポートされます。未サポートの組み込み関数や構造体、または共用体の配列のクロスセクションでは、予期せぬ結果をもたらす可能性があります。製品マニュアルの“リファレンス＞メインフレーム リファレンス＞Open PL/I リファレンス＞ Open PL/I 言語リファレンス マニュアル＞Open PL/I の組み込み関数”で内容をご確認ください。

④ ¥付きの変数またはラベル

¥または Linux 上のバックスラッシュを使用した変数名やラベル名は、PL/I 識別子として認められていないためエラーになります。

8.3 コンパイラオプション指定の注意点

PL/I コンパイラオプションを使用する際の注意点と、メインフレームと実装形式の違いにより発生する代表的な非互換を説明します。

① 制御指定：systemcics、systemims、systemmvs コンパイラオプション

OPTIONS（MAIN）プログラムが各ミドルウェア配下で制御されるように指定するものです。個別にプログラムをコンパイルする場合はメインプログラムだけに指定してください。

② 外部ファイルを使用する非標準文字の使用：defext コンパイラオプション

この指定をしたプログラムにおいては、STATIC EXTERNAL 変数と外部ファイル定数の初期値が定義されるため、サブルーチンとして呼ばれる複数のプログラムに重ねて定義することができません。1つのプログラムだけに外部ファイルを定義するなどの対策が必要になります。

③ エンディアン：bigendian コンパイラオプション

デフォルトは、バイナリ数値のバイトオーダーをターゲットプラットフォームの CPU 固有のバイトオーダーに合わせます。このため Intel CPU ではリトルエンディアンと解釈して最適化された高速なコードを生成しますが、メインフレームとはエンディアンが異なります。このオプションを指定すると、データ項目に対して NATIVE 属性が明示的に適用されていない限り、すべての FIXED BINARY、CHARACTER VARYING、GRAPHIC VARYING、および WIDECHAR VARYING 項目は、ビッグエンディアンと解釈したコードを生成します。これによってメインフレームと互換性のあるデータ形式となりますが、実行時に若干のオーバーヘッドが発生します。

④ 浮動小数点の桁数：zfloat コンパイラオプション

BIN FLOAT 型において、IBM メインフレームでは最大 53 桁に対し、Enterprise Developer では 52 桁とされています。これは、オープン系プラットフォームでは浮動小数点数が IEEE 方式で扱われていることに起因するもので、メインフレームとのマシンアーキテクチャの相違による制限事項となります。例えばメインフレームとの互換性を重視して-zfloat コンパイラオプションを使用した場合、BINARY FLOAT（22）の変数が IEEE 仕様の浮動小数点形式では 4 バイトで格納可能であるにもかかわらず 8 バイトを要するようになります。リプラットフォームにあたり、8 バイトを要する箇所だけを BINARY FLOAT(23)に書き直すという方法を取ることで、より最適化されたコードで実行できるようになります。

zfloat コンパイラオプション）

浮動バイナリ単精度のデフォルトを 21 に、浮動バイナリ倍精度の最大値を 53 に指定します。デフォルトは、それぞれ 23 および 52 です。

⑤ DBCS サポート：graphic コンパイラオプション

このオプションを指定することで GRAPHIC データ型のサポートを有効にします。ただし、-ebcdic コンパイラオプションを指定する場合は WCHAR および GRAPHIC データ型はサポートされませんので、併用は避けてください。

ebcdic コンパイラオプション）

すべての文字データに対して、EBCDIC 文字エンコードを使用することを指定します。

⑥ ASCII/SJIS モードにおける EBCDIC サポート

ASCII/SJIS 文字コードデータを扱うモードを選択時に、EBCDIC 順のソートや比較を行うことはできません。必要ば場合は、独自で組み込み関数を用意するなどの対策が必要になります。

その他の制限事項については、製品マニュアルの “リファレンス＞メインフレーム リファレンス＞Open PL/I リファレンス＞Open PL/I 言語リファレンスマニュアル”の各項目で内容をご確認ください。

8.4 生成されるファイル

コンパイルやリンクの過程で生成されるファイルを拡張子別に説明します。一部のファイルはコンパイルオプションにより生成されない場合もあります。

① 実行モジュール

- .dll … Windows 環境の実行モジュールです。
- .so … Linux/UNIX 環境の実行モジュールです。

② デバッグ関連ファイル

- .stb … 独自デバッガである CodeWatch が使用するデバッグ情報ファイルです。
- .pdb … Windows 環境で PLIDUMP を使用時に必要となるデバッグ情報ファイルです。

③ 中間ファイル

- .obj … Windows 環境のオブジェクトモジュールです。
- .o … Linux 環境のオブジェクトモジュールです。
- .def … リンク定義ファイルです。
- .exp … エクスポートされる関数やデータの情報ファイルです。
- .lib … インポートライブラリファイルです。
- .dcf … デバッグ用データ収集ファイルです。

④ データファイルツール関連ファイル

・ .adt … データファイルツールでレコードレイアウトを生成する際に使用するファイルです。

9. COBOL プログラムと PL/I プログラム間の呼び出し

COBOL と PL/I の実行モジュールは互いに呼び合うことができます。この方法と注意点を説明します。

9.1 AMODE コンパイラ指令

PL/I は、COBOL の AMODE(31)コンパイラ指令に相当するメインフレーム形式のポインタマッピングをサポートしておらず、常にネイティブ OS の実アドレスとなっています。そのため、PL/I と COBOL を混在させる場合、COBOL プログラムのコンパイル時に AMODE(31)や AMODE(24)を指定すると正しく動作しません。これは製品設計上の制限事項となります。

9.2 COBOL メインプログラムから PL/I サブルーチンの呼出し

COBOL アプリケーションの制御下で実行される PL/I サブルーチンを使用するには、PL/I プログラムを呼び出す前に PL/I ランタイムを初期化し、終了前にシャットダウンするコーディングが必要となります。

PL/I ランタイムの初期化コーディングの例)

```
special-names.
    call-convention 8 is litlink.

     *> PL/I ランタイムの初期化
     call litlink '_lpi_init' using by value 0 size 4
                                    by reference argv
                                    by reference argv
                                    by value pli-lang
     *> PL/I サブルーチンの呼出し
     call 'PLISUB'
     *> PL/I ランタイムの終了
     call litlink '_lpi_fini_and_return' using
                                    by value pli-lang
                                    by reference pli-retcode
```

詳しくは製品マニュアルの“プログラミング>メインフレームプログラミング>PL/I プログラミング>Open PL/I ユーザーガイド>Open PL/I の使用>他の言語からの PL/I の呼び出し”をご参照ください。

また、PL/I ランタイムの初期化とシャットダウンを使用するためには、リンク時にライブラリの指定が必要です。
IDE を利用時はプロパティ設定の画面にて[追加のリンクファイル]へライブラリを指定します。

Windows ライブラリ) mfplimd.lib
Linux ライブラリ) lmfpliz

補足)
PL/I メインプログラムから COBOL サブルーチンを呼出し、さらに PL/I サブルーチンを呼び出している場合は、既にメインプログラムにて PL/I ランタイムを使用していることから、初期化とシャットダウンのコーディングは必要ありません。

① COBOL と PL/I の静的リンク実行モジュールを生成する場合

Windows のコマンド例)mfplimd.lib

```
cobol CBLMAIN.cbl;
mfplx -c PLISUB.pli -o PLISUB.obj
cbllink CBLMAIN.obj PLISUB.obj mfplimd.lib
```

Linux のコマンド例)-L$COBDIR/lib -lmfpliz64

```
cob -xc CBLMAIN.cbl
mfplx -c -pic PLISUB.pli -o PLISUB.o
cob -z CBLMAIN.o PLISUB.o -L$COBDIR/lib -lmfpliz64
```

② COBOL と PL/I の動的ロード実行モジュールを生成する場合

Windows のコマンド例)mfplimd.lib

```
cbllink CBLMAIN.cbl mfplimd.lib
mfplx -dll PLISUB.pli
```

Linux のコマンド例)-L$COBDIR/lib -lmfpliz64

```
cob CBLMAIN.cbl -U -o CBLMAIN.so -L$COBDIR/lib -lmfpliz64
mfplx -dll PLISUB.pli -o PLISUB.so
```

9.3 PL/I メインプログラムから COBOL サブルーチンの呼出し

PL/I ランタイムの初期化とシャットダウンのコーディングの追加は必要ありませんが、内部的にライブラリを使用するため、前項と同様にライブラリの指定が必要です。

① PL/I と COBOL の静的リンク実行モジュールを生成する場合

PL/I プログラムソースの ENTRY 属性に、静的リンクによって COBOL プログラムを呼び出すことを明示的に記述します。

PL/I のコード例）

```
DCL CBLSUB ENTRY OPTIONS(COBOL);
```

Windows のコマンド例)mfplimd.lib）

```
mfplx -c PLIMAIN.pli -o PLIMAIN.obj
cobol CBLSUB.cbl;
cbllink PLIMAIN.obj CBLSUB.obj mfplimd.lib
```

Linux のコマンド例)-L$COBDIR/lib -lmfpliz64）

```
mfplx -c -pic PLIMAIN.pli -o PLIMAIN.o
cob -xc CBLSUB.cbl
cob -z PLIMAIN.o CBLSUB.o -L$COBDIR/lib -lmfpliz64
```

② PL/I と COBOL の動的ロード実行モジュールを生成する場合

PL/I プログラムソースの ENTRY 属性に、動的ロードによって COBOL プログラムを呼び出すことを明示的に記述します。

PL/I のコード例）

```
DCL CBLSUB ENTRY OPTIONS(FETCHABLE,COBOL);
```

Windows のコマンド例)mfplimd.lib）

```
mfplx PLIMAIN.pli
cbllink –d CBLSUB.cbl mfplimd.lib
```

Linux のコマンド例)-L$COBDIR/lib -lmfpliz64）

```
mfplx PLIMAIN.pli -o PLIMAIN.so
cob –u CBLSUB.cbl -o CBLSUB.so -L$COBDIR/lib -lmfpliz64
```

10. Enterprise Developer チュートリアルと例題

Enterprise Developer の製品マニュアルには、PL/I ユーザーに向けた JCL、CICS、IMS などのチュートリアルを準備しています。これらのチュートリアルには IDE を使用したプロジェクトの作成方法からコンパイル、実行、デバッグまでを具体的に記述しており、例題もダウンロードできますので、ぜひご参照ください。

メインフレーム チュートリアル

このセクションには、メインフレーム上での開発を対象としたチュートリアル (CICS、IMS、JCL、および Open PL/I アプリケーションの開発方法など) が含まれています。

Enterprise Developer - PL/I JCL チュートリアル
このチュートリアルでは Enterprise Developer for Eclipse による PL/I 言語の JCL アプリケーション開発および移行方法を案内します。

Enterprise Developer - リモート開発チュートリアル
このチュートリアルのセットは、Enterprise Developer for Eclipse による Linux リモート開発方法を案内します。

Enterprise Developer - CICS システム間通信チュートリアル
このチュートリアルでは Enterprise Developer for Eclipse を使用して CICS システム間通信を実施する方法を案内します。

Enterprise Developer - PL/I CICS チュートリアル
このチュートリアルでは Enterprise Developer for Eclipse による PL/I 言語の CICS アプリケーションの開発および移行方法を案内します。

11. おわりに

IBM メインフレーム環境では資源を開発者全員で共有するために、急を要するコンパイルは優先度の調整を行う、変数の値を確認するためにデバッグ文を入れたプログラムを再コンパイルする、などの開発スタイルが多く見受けられます。一方、オープン環境の Enterprise Developer は Eclipse, Visual Studio といった業界標準の IDE にアドオンする形で PL/I アプリケーションの開発ができ、Visual Studio Code では PL/I 拡張機能をインストールすることで PL/I の開発作業をより軽く迅速に行うことができます。

ステップ実行を利用したデバッグ、変数値の動的な確認、処理の流れの動的な把握など、他開発言語と同様の開発スタイルで、効率的な開発環境を整えることができる製品です。
また、開発用実行環境の Enterprise Server インスタンスを開発者ごとに占有することができるため、コンパイルや実行時に他開発者との調整は必要なく、これによる開発工数の削減も見込めます。

IBM メインフレーム環境との違いおよび注意点をご理解いただき、Enterprise Developer を活用したリプラットフォームをご検討いただければ幸いです。

補足:稼働環境

本書は下記環境を基に記述しています。

1）OS

Windows 11 Pro

2）プロセッサ

13th Gen Intel® 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）使用した製品マニュアル バージョン

Enterprise Developer 11.0 for Eclipse