Rocket software

2025/12/11 更新

Visual COBOL アプリケーション互換性ガイド

Visual COBOL 11.0 製品への移行

アプリケーションのポータビリティ（移植可能性）はプログラミング言語にとって永遠の課題と言えます。コンパイラのバージョンアップに伴ってプログラムがコンパイルエラーになる、あるいは動作が変わるという事象は過去 50 年間以上に及ぶプログラミング言語の歴史の中で幾度となく繰り返されてきました。

その中でも COBOL はもっともポータビリティに優れた言語であり、言語規格自体もポータビリティを保ちつつ更新されてきました。この COBOL 言語を扱う COBOL 製品も同様にポータビリティの維持を意識して 40 年以上に渡り開発されています。

しかし、このように言語や製品側で高いポータビリティを保っていても、プログラムの性質や関連するハードやミドルウェア等がからみ動作環境によっては規格や製品が提供する互換設計だけではカバーできないものもあります。本書では、言語としての例外ケース、製品のバージョンアップ／変更や OS 及び関連ミドルウェアの変更等に伴う注意事項等をそれぞれ整理して解説します。

なお、本書はお客様がフィールドで検出された動作の違いをフィードバックとして随時改版されています。そのため、網羅性を保証するものではないことをご理解の上ご活用いただきますとともに、常に最新バージョンを参照していただくことをお勧めいたします。

目次

1. はじめに

アプリケーションのポータビリティ（移植可能性）はプログラミング言語にとって永遠の課題と言ってもよいでしょう。コンパイラのバージョンアップに伴ってプログラムがコンパイルエラーになる、あるいは動作が変わるという事象は過去 50 年間以上に及ぶプログラミング言語の歴史の中で幾度となく繰り返されてきました。
その中で COBOL はもっともポータビリティに優れた言語であると言えます。その主な理由を以下に列挙します。

a) 国際規格で保護された言語仕様
Java はプラットフォーム間の互換性は非常に高いです。ただしその一方で、バージョン間の互換性に関しては完全とは言えず、Java 1.1 から 1.3 へのバージョンアップでは過去多くの互換性問題が発生しました。Visual Basic もまた然りです。Visual Basic 6 から Visual Basic .NET(VB 7) へのバージョンアップについては、後方互換がなく、変換ツールを使っても完全に変換ができたというケースも稀でした。
COBOL については 1977, 1985, 2002 と旧規格からの互換性を入念に考慮した精緻なバージョンアップが図られてきました。

b) 抽象度の高い言語仕様
ファイル割り当てやデータベース接続などの環境依存部分をできる限りソースコードの外部にくくりだすコーディングが可能です。これにより、環境が変わっても少量のソース変更で移植ができます。

c) ハードウェアへの非依存
COBOL はバイトエンディアン、32-bit/64-bit などのハードウェア属性にとらわれないコーディングが可能です。

さらに Visual COBOL は OS 非依存のライブラリルーチンを提供するなど、移植可能性を意識した機能を多く装備します。

しかしながら、サーバーリプレースなどに伴うバージョンアップやプラットフォーム変更で、COBOL アプリケーションのポータビリティについて過度な期待を持つことは危険です。
本書では、様々なケースについて COBOL アプリケーションの互換性について考慮すべき点を整理して解説します。

2. 同じプラットフォームでのバージョンアップ

本章では、サーバーリプレースなどに伴い同じプラットフォームの新しい OS バージョンや COBOL およびミドルウェア製品のバージョンの環境に移行する際に考慮すべき注意点について述べます。尚、本章で述べる注意点の多くはプラットフォームをまたいだ移行の際にも注意いただくべき内容となります。

2.1. コンパイルチェックに関する非互換の発生要因

Visual COBOL の COBOL コンパイラに限らず主なコンパイラは、フロントエンドにてプログラムをバックエンドが使える仕様に合うよう加工します。その際、字句解析、構文解析、意味解析等のフェーズを経て検出するプログラムソース中の誤りをエラーや警告としてユーザーに通知し言語規約に違反した不正なアプリケーションが生成されることを可能な限り抑止します。このエラーハンドリング機能を含め、バージョンを重ねるごとに実行速度の向上、実行時の消費資源の削減（メモリ、CPU など）の改善が Visual COBOL の COBOL コンパイラには加えられています。このエラーハンドリング機能の機能強化により過去のバージョンや製品では検出されなかった不正なコードがフィルターされ、より厳密に規約に準じたコードに匡正することができます。また逆に、機能の拡張により解釈できる範囲が拡がり、過去バージョンや製品ではエラーと解釈されていたものが正しいコードと解釈されるものもあります。[1]

1 COBOL コンパイラの改善・改修により顕在化した主な事象を付表 1 にまとめています。

2.2. 動作非互換の発生要因

Visual COBOL の COBOL コンパイラは、長年にわたって旧バージョンでサポートされてきた COBOL 構文を継続してサポートし続けてきました。このため正しい意図のもとに書かれている COBOL プログラムは上位バージョンのコンパイラでも同様にコンパイルして実行することができます。ただし以下のような場合にはそうではない場合もあります。

- コンパイラ指令・ランタイム構成・ファイルハンドラー構成の相違
- 実行形式の相違
- COBOL 言語仕様として動作が規定されていない構文[2]
- 実行時のデータとして結果が規定されないもの [2]
- コンパイラやランタイムの機能拡張に伴い優先される機能の変更[3]

2.3. 廃止機能に伴うコンパイル設定、実行時構成、運用コマンド等の変更

COBOL 製品は旧バージョンとの互換性を保てるよう基本的に旧バージョン、旧製品で指定するコンパイルの命令や実行時構成は引き継げるよう製品設計しています。指定するオプションについてもオプションを追加することはあっても既存のオプションの廃止や、内容変更することは基本的にありません。しかし、提供機能を見直す際、サポートプラットフォームの進化に伴いフィットしない機能や重複する機能等は廃止し、より利便性を高める機能の開発に注力することもあります。本項では主に旧製品 Net Express 及び Server Express から Visual COBOL の最新版のリリースに至るまでの間に廃止となったコンパイラ指令[4][5]、実行時チューナー、ファイルハンドラー構成オプション、コマンドラインシンタックス等、設定や運用コマンドに影響しうる廃止・変更要素を列挙します。これらを現行のコンパイルシェル、実行時シェル、プログラム等で利用している場合は、移行後におけるこれらの機能の要否等を確認します。ただ、これらは上述のような背景で廃止しているため、多くのケースでは仮に現行利用していても移行後のプラットフォームでは不要となる機能が大半となります。「*」マークが付いたものは指定をせずともその機能が有効になる、或いは別の指令 / 機能にて引き継がれたオプションとなります。廃止以後のバージョンを利用しているのであれば廃止機能に関する考慮は不要です。尚、製品内部使用のオプションは本書の対象から除いています。

- 廃止されたコンパイラ指令 (Windows)
  - ANIMPREP ・・・ Visual COBOL 2.1 にて廃止
  - ANSI2000* ・・・ Net Express 5.1 にて廃止 (ISO2002 指令で代替)
  - EDITOR ・・・ Visual COBOL 2.1 にて廃止
  - FAULTFND ・・・ Visual COBOL 2.2 にて廃止
  - FLAGCD ・・・ Visual COBOL 2.1 にて廃止
  - KEEP-INT ・・・ Visual COBOL 2.1 にて廃止
  - NEWBASENAME ・・・ Visual COBOL 2.1 にて廃止

2 主なケースの詳細は 5 章で述べています。

3 Visual COBOL は旧製品、バージョンからの移行性を意識して製品設計されているため、多くのケースではこの変更による影響は被りません。逆に拡張機能による改善を意識することなく享受できるケースが大半です。しかし、特殊な運用やコーディングをしている場合は対応が必要となることもありますため、付表 2 でまとめた主な変更点については少なくとも確認してください。

4 ネイティブ開発向けの指令を対象としています。

5 旧製品 Net Express 及び Server Express の各リファレンスのコンパイラ指令一覧に記載されていない指令に関しては、これらの製品よりもさらに古い COBOL 製品で使用されていたものである可能性がありますが、そのような旧製品の指令は基本的にサポートされません。

> 廃止されたコンパイラ指令 (Linux/UNIX)

- ANIMPREP ・・・ Visual COBOL 2.1 にて廃止
- ANSI2000* ・・・ Server Express 5.1 にて廃止 (ISO2002 指令で代替)
- CONVERTPTR* ・・・ Server Express 4.0 にて廃止 (AMODE 指令で代替)
- EDITOR ・・・ Visual COBOL 2.1 にて廃止
- FAULTFND ・・・ Visual COBOL 2.2 にて廃止
- FLAGCD ・・・ Visual COBOL 2.1 にて廃止
- ISO2000* ・・・ Server Express 4.0 にて廃止 (ISO2002 指令で代替)
- KEEP-INT ・・・ Visual COBOL 2.1 にて廃止
- NESTCALL* ・・・ Server Express 4.0 にて廃止 (NEST プログラムを標準サポート)
- NEWBASENAME ・・・ Visual COBOL 2.1 にて廃止
- RDEFPTR* ・・・ Server Express 4.0 にて廃止 (AMODE 指令で代替)
- RNIM ・・・ Server Express 4.0 にて廃止

> 16 ビット版製品でのみ有効なコンパイラ指令

- 01SHUFFLE
- 64KPARA
- 64KSECT
- AUXOPT
- CHIP
- DATALIT
- EANIM
- EXPANDDATA
- FIXING
- FLAG-CHIP
- MASM
- MODEL
- OPTSIZE
- OPTSPEED
- PARAS
- PROTMODE
- REGPARM
- SEGCROSS
- SEGSIZE
- SIGNCOMPARE
- SMALLDD
- TABLESEGCROSS
- TRICKLECHECK

> 廃止された実行時チューナー（Windows）

- dynamic_memory_limit* ・・・ Net Express 5.0 にて廃止 (default_cancel_mode で代替)
- faultfind_config ・・・ Visual COBOL 2.2 にて廃止
- faultfind_level ・・・ Visual COBOL 2.2 にて廃止
- faultfind_outfile ・・・ Visual COBOL 2.2 にて廃止
- faultfind_recsize ・・・ Visual COBOL 2.2 にて廃止
- isam_block_size* ・・・ Net Express 5.0 にて廃止（NODESIZE で代替）
- isam_open_key_check* ・・・ Net Express 5.0 にて廃止（KEYCHECK で代替）
- program_search_intgnt* ・・・ Visual COBOL 2.1 にて廃止（COBPATH 環境変数で代替）
- program_search_order ・・・ Visual COBOL 2.1 にて廃止
- skip_on_lock* ・・・ Net Express 5.0 にて廃止（SKIPLOCK で代替）
- win9x_shiftlock_fixd* ・・・ Visual COBOL 2.1 にて廃止（Windows 9x 向け機能）

> 廃止された実行時チューナー（Linux/UNIX）

- bll_cell_address_check ・・・ Server Express 4.0 にて廃止
- detect_alt_ctrl* ・・・ Server Express 4.0 にて廃止 (SCO Open Desktop 2 より前の環境向け機能)
- faultfind_config ・・・ Visual COBOL 2.2 にて廃止
- faultfind_level ・・・ Visual COBOL 2.2 にて廃止
- faultfind_outfile ・・・ Visual COBOL 2.2 にて廃止
- faultfind_recsize ・・・ Visual COBOL 2.2 にて廃止
- long_filenames* ・・・ Server Express 4.0 にて廃止 (255 文字までのファイル名を標準サポート)
- multi_close_limit* ・・・ Server Express 4.0 にて廃止（制限自体を撤廃）
- pc_mono_palette ・・・ Server Express 4.0 にて廃止
- program_search_intgnt* ・・・ Visual COBOL 2.1 にて廃止 (COBPATH 環境変数で代替)
- program_search_order ・・・ Visual COBOL 2.1 にて廃止
- redir_stdin_is_recsq ・・・ Visual COBOL 2.3 にて廃止
- signal_interface ・・・ Server Express 4.0 にて廃止 (旧製品との互換機能)

> 廃止されたファイルハンドラー構成オプション（Windows、Linux/UNIX）

- USEVSAMKEYDEFS ・・・ Visual COBOL 2.2 にて廃止 (上位製品で継続サポート)

> 廃止された環境設定用スクリプト（Linux/UNIX）

- cobsje ・・・ 廃止。 Visual COBOL では $COBDIR/bin/cobsetenv にて Java 連携に必要な CLASSPATH 等も併せて設定

> 変更・廃止されたコマンド／コマンドラインシンタックス (Windows)

- cobolc、cobolw コマンド
  Net Express ではキャラクターモードで実行するアプリケーションをコンパイルする際は cobolc.exe を、グラフィックモードで実行するアプリケーションをコンパイルする際には cobolw.exe を利用していました。Visual COBOL ではこれらを統一して全てのアプリケーションを cobol.exe でコンパイルできるよう改善しています。

- runc、runmc コマンド
  Net Express ではキャラクターモードとして運用するアプリケーションを実行する際は、runc.exe もしくは runmc.exe を利用していました。Visual COBOL ではそれぞれ run.exe、runm.exe で代替できます。

- runs、runsc、runsw コマンド
  Net Express ではシングルスレッドランタイムが runs.exe、runsc.exe、runsw.exe として提供されていました。Visual COBOL ではシングルスレッドアプリケーション、マルチスレッドアプリケーション問わずマルチスレッドランタイムで実行されます。runs* のうち「s」を除いたコマンドもしくは「s」を「m」に置き換えたコマンドで実行します。

- cbllink コマンド
  - -b フラグを廃止 ・・・ Visual COBOL では静的ランタイムシステムへのリンクは不要となっています。
  - -w フラグを追加 ・・・ システムリンカーや C コンパイラへ情報を渡せるようになりました。
  - -y フラグを追加 ・・・ Visual COBOL にて Windows XP や Windows Server 2003 をターゲットとしたアプリケーションを開発する場合に指定します。
  - -s フラグを廃止 ・・・ Visual COBOL ではシステムモジュールへのリンクは不要となっています。

- projmake コマンド
  本コマンドは Net Express IDE で作成するプロジェクトに対して有効なコマンドです。Visual COBOL では Net Express IDE は使用しないため、本コマンドは使用できません。

- mfnetx コマンド
  本コマンドは Net Express IDE で作成するプロジェクトを開くためのコマンドです。上述の理由により本コマンドは Visual COBOL では使用できません。

> 変更・廃止されたコマンドラインシンタックス (Linux/UNIX)

本書執筆時点では特になし。
Visual COBOL の 1 つ前の世代の製品 Server Express ではコマンド、ビルドコマンドとして cob コマンドが用意されていました。本コマンドは Visual COBOL でもサポートされており、各コマンドフラグも同義の効果を継続維持したままサポートされています。フラグについては Visual COBOL では下記のようなものも加えられました。

- +CC cc_option ・・・ C++ コンパイラへオプションを渡します。
- -j ・・・ COBOL を Java bytecode へコンパイルします。
- -y[,U][,CC] ・・・ 従来の -y のみのフラグに加え、未定義のシンボルがある場合にエラーメッセージ出力させる U 並びに C++ のオブジェクトをライブラリへ加える CC も指定できるようになりました。

> 変更されたコマンドラインシンタックス(Windows、Linux/UNIX 共通)
imtkmake コマンド
サポートされる Java EE アプリケーションサーバー及び規格がアップデートされていますため、下記コマンドオプションについては少なくとも新環境に合わせて見直す必要があります。

- imtkmake -generate [appserver]・・・[j2eeVersion]
- imtkmake -queryAppserverList [j2eeVersion]
- imtkmake -genclient [appserver]・・・[j2eeVersion]

また、同コマンドは随時強化されており、コマンドオプションもバージョンアップの度に追加されています。これらの追加オプションの指定要否についてもバージョンアップに際して確認します。

> 変更された環境変数の解釈(Windows)
COBDIR
Net Express 以前の Windows 版の旧製品では PATH 環境変数に加え、COBDIR 環境変数にもアプリケーション配置先の各フォルダや製品のバイナリフォルダ(< システムフォルダ >¥bin 等) を設定することが可能でした。Visual COBOL では COBDIR 環境変数へは製品システムフォルダ（インストールフォルダ）を設定します。

2.4. その他の廃止機能

本項では、2.3 で紹介した構成や運用コマンドに影響し得る要素に加えて Net Express 並びに Server Express より廃止となった機能を列挙します。これらも前項と同様ユーザーやマーケット需要を鑑みて見直した機能となりますため、Visual COBOL への移行後は考慮不要となるケースが大半となります。

> コンパイル、ビルド関連 (Windows)
- 静的ランタイム (Visual COBOL では cbllink –b によるリンクや、Net Express プロジェクトにおけるリンク構成の実行時ライブラリオプション「静的」選択に相当する設定が不要となります。)
- シングルスレッドランタイム (シングルスレッドアプリケーション、マルチスレッドアプリケーションともにマルチスレッドランタイムシステムで動作します。)
- COM 作成ウィザード (Net Express IDE に付属していた COM 作成ウィザードは廃止となりましたが、COM のランタイムは継続して提供されます。そのため、COM 作成ウィザードで作成したソースを Visual COBOL でコンパイルし、運用することは可能です。)

> 廃止されたライブラリルーチン (Windows)
- PC_ISAP_GET_EXT ・・・ Visual COBOL 2.1 にて廃止

> 廃止されたライブラリルーチン (Linux/UNIX)
- CBL_DIR_SCAN_BEGIN* ・・・ Server Express 4.0 にて廃止
(CBL_DIR_SCAN_START で代替)
- CBL_FFND_REPORT ・・・ Server Express 4.0 にて廃止
- X"91" function 52* ・・・ Server Express 4.0 にて廃止 (LSRECDELIM 等で代替)
- X"91" function 53* ・・・ Server Express 4.0 にて廃止 (LSRECDELIM 等で代替)

> 廃止されたユーティリティ
- animserv によるリモートデバッグ (Visual COBOL ではこれに取って代わるより高機能なリモートデバッグ機能を提供しています。)
- CBL2XML (コマンドライン版は引き続きサポートされます。)
- cobimtk (cobimtk で実現していた GUI による Interface Mapping Toolkit 機能は Visual Studio 及び Eclipse IDE 上に取り込み、IDE における COBOL プロジェクトとの連携を可能にしました。)
- Form Designer (Net Express 付属のユーティリティ)
- Faultfinder
- FSView (コマンドライン版は引き続きサポートされます。)
- GNT Analyzer
- ISAPI
- NSAPI
- On-line Help Builder (Net Express 付属のユーティリティ)
- OO Class and Method ウィザード (Net Express 付属のユーティリティ)
- OpenESQL の OCI サポート (ODBC, ADO.NET は引き続きサポートされます。)
- Solo Web Server (Net Express に付属する CGI ベースの Web アプリケーションを開発する機能)
- Type Library Assistant (Net Express 付属の COM 関連ユーティリティ)

2.5. 製品付属ユーティリティの機能拡張及び改善

Visual COBOL にはファイル処理、パフォーマンス分析、カバレッジ分析等といった機能を提供するユーティリティを豊富に備えます。これらの多くは旧製品より提供されていたものを引き継いだものです。これらのユーティリティに対しても旧製品、旧バージョンより機能が弱い部分は強化され、障害があれば改修を重ねてきました。その結果、Visual COBOL の最新版にバージョンアップしたタイミングで、機能強化や障害改修が施されたことにより問題が顕在化されることもあれば、障害による誤動作を前提に運用している場合は機能是正により挙動が旧製品・旧バージョンと異なるようになることもあり得ます。[6]

2.6. OS, ミドルウェア製品のバージョン相違

COBOL ランタイムシステムはターゲット環境の OS が提供するシステム API を使用して動作します。このため OS のバージョン間非互換の影響を受ける可能性がゼロではありません。ただし、COBOL が利用する多くの API は OS 自身が互換を保証する標準的なものとなります。

アプリケーションが RDB 等のミドルウェアと連携し、そのミドルウェアもバージョンアップする場合、Visual COBOL 側だけでなくミドルウェア自体のバージョン間の差分も確認する必要があります。バージョンアップに伴うミドルウェアの作りこみによりミドルウェア自体の挙動や構成方法が変わる場合もあります。[7] Visual COBOL 側については以下のような観点で確認します。

> WebLogic, WebSphere 等の Java EE Application Server のバージョンアップ
- COBOL サービスを構成する際に指定する Java EE Application Server のバージョンをバージョンアップ後のものに合わせて再ディプロイ。
- Java EE Application Server に組み込むリソースアダプターを新しいバージョンに合わせて組み込み。(Visual COBOL は Java EE Application Server の種類別、バージョン別にリソースアダプターを用意しています。)

> RDB のバージョンアップ (XA を使って Enterprise Server を運用する場合)
新しい環境に合わせた XA スイッチモジュールを再作成。

6 これまでに報告された主な事象を付表 3 にまとめています。

7 弊社側で確認できたミドルウェアをバージョンアップする際の留意点を付表 4 に記します。ただし、こちらはあくまでも第 3 者、もしくはパートナーの立場として確認した例も含まれております。これらは弊社側で完全性は保証できるものではないため、必要に応じて各ミドルウェアの提供ベンダーへお問い合わせをお願いします

- RDB のバージョンアップ (OpenESQL を利用して RDB へアクセスする場合[8])
  OpenESQL オプション TARGETDB の見直し。

- Oracle のバージョンアップ (COBSQL を利用する場合)
  COBSQL プリプロセッサオプション COBSQLTYPE の見直し。

- Db2 のバージョンアップ (DB2 ECM を利用して RDB へアクセスする場合)
  DB2 指令オプション UDB-VERSION の見直し。

2.7. OS が提供する基本ライブラリによる影響

Visual COBOL 製品をはじめ、全てのアプリケーションは、OS が提供する基本ライブラリを利用して動作します。Red Hat Enterprise Linux 10 のように 32-bit 版をサポートしない OS 上では、弊社製品に関わらず、全ての 32-bit アプリケーションが動作しません。このような OS に 32-bit アプリケーションを移行する際は 64-bit への移行が必要となります。
留意すべき点につきましては、4.1 をご参照ください。

8 Visual COBOL は、firehose カーソル接続やプリフェッチ等を調整してカーソル処理の最適化を図る BEHAVIOR オプションを追加しました。デフォルトでは最適化パフォーマンスを導出するためのオプションが有効となっていますが、カーソル処理の内部動作も旧製品と互換性を保たせたい場合は、無効化オプション UNOPTIMIZED を BEHAVIOR 指令に指定します。

3. プラットフォームの変更を伴う移行

本章では、サーバーリプレースに際してサーバーアーキテクチャーの変更を伴うケースにて考慮すべき問題点について述べます。このケースでも 2 章で述べた点はすべて考慮に値します。ここではさらに追加で考慮が必要となる点について論じます。

3.1. Windows と Linux/UNIX の相違

COBOL は OS 間の移植性の高い言語と言われています。それであっても以下に列挙するような環境に依存する部分については移行にあたって配慮が必要となります。

- OS 固有機能の呼び出し
  Linux/UNIX のシステムコールや Win32 API を COBOL から CALL するアプリケーションを Visual COBOL で開発できます。Windows – Linux/UNIX をまたがる移行の際はこれらの CALL は見直す必要があります。

- パス名のハードコーディング
  ファイルの割り当ての際等にパス名をソース中にハードコードしている場合は、移行先が同じパスで構成されていない限り、参照することができません。また Windows 環境ではフォルダの区切りに「¥」(円マーク、英語 OS ではバックスラッシュ) を使い、Linux/UNIX 環境では「/」を使用します。Windows 環境であっても、COBOL プログラムは「/」をフォルダの区切りとして認識するため、ハードコードする必要があるのであれば、Windows 環境でも「/」で記述します。

- ケースセンシティビティ
  Windows のファイルシステムは大文字・小文字を区別しませんが、Linux/UNIX 環境のファイルシステムは大文字・小文字を区別します。そのため、移行元が Windows でプログラム中に記述するサブプログラム名、エントリ名、ファイル名の大文字・小文字が意識されていない場合は、Linux/UNIX の環境に合わせて確認する必要があります。

- ファイルの改行文字
  Windows のファイルシステムでは CRLF(X'0D0A') を改行文字として認識し、Linux/UNIX では LF(X'0A') として認識します。例えば、固定長相対ファイルのレコード区切りはファイルハンドラー構成オプション RELRECDELIM で指定しますが、これが環境と矛盾がないか確認する必要があります。本オプションは Windows 環境版では 0D0A が、Linux/UNIX 環境版では 0A がデフォルト値となるため、移行する構成ファイル中で本オプションが構成されていない場合、意識する必要はありません。

- ライブラリルーチン
  CBL_ をライブラリルーチン名の先頭に持つライブラリルーチンは Windows – Linux/UNIX 間のポータビリティが保証されています。しかし、その他の X"91" function 11 のようなルーチンに関してはポータビリティが保証されていません。

- 拡張文字セット
  Windows 環境は IBM の拡張図形文字セットのような拡張文字セットをサポートしますが、Linux/UNIX 環境ではサポートされないものもあります。CBL_GET_SCR_LINE_DRAW や CBL_GET_SCR_GRAPHICS ルーチンを利用し、環境にとらわれないコーディングをして回避もできます。

- Alt キー、Ctrl キー
  Linux/UNIX 環境の多くは Alt キー、 Ctrl キーが単体で認識されません。Windows 環境で構築したアプリケーションがこれらのキーの使用を前提としていましたら、Linux/UNIX 環境への移行の際は注意が必要です。

3.2. バイナリ数値のエンディアンの相違

COBOL には数値をバイナリ形式で保持するデータタイプが幾つか用意されています。これらのバイトオーダーは下記のような特色があるため、異なるエンディアンを持つシステム間の移行に際して注意が必要となる場合があります。

> OS のエンディアンに合わせたバイトオーダー

対象となるデータ形式	USAGE COMPUTATIONAL-5
特徴	・ パフォーマンスに最も優れた形式 ・ エンディアンが異なる OS 間ではデータ互換性なし → REDEFINES などで参照している場合は挙動が変わる可能性もあり

> ビッグエンディアン

対象となるデータ形式	・ USAGE COMPUTATIONAL ・ USAGE BINARY ・ USAGE COMPUTATIONAL-4 ・ USAGE COMPUTATIONAL-X
特徴	OS のエンディアンに関わらず、固定でビッグエンディアン形式で保持

3.3. プラットフォーム間の文字コードの相違

Visual COBOL は各種日本語の文字コードでエンコードされたソースやデータファイル等を扱えます[9]。これらのコードはプラットフォーム間で正しく情報交換できるよう国際規格等によって標準化が進められています。しかし、各プラットフォームでロケールとして提供される規則は独自に拡張されていることもあるため、注意が必要です。

例えば、各 OS で SJIS として利用できるロケール／文字コードでもそれぞれで仕様が異なることもあるようです。そのため、JIS X0208 のような標準規格にはない特殊な文字を扱う際は移行先のプラットフォームでそれらの文字を正しく扱うよう環境を整備した上で開発に着手する必要があります。Visual COBOL のコンパイラやランタイムは iconv 等 OS が提供するロケールに依存した関数を使用し、コンパイル時にソース中の２バイト文字の解釈や実行時に２バイト文字操作等をする機能を提供します。OS 上でこれらの関数を正しく扱えるよう構成していないと、コンパイルエラーや実行時に文字化け等の事象を引き起こす可能性があります。

下表は一部の環境にて確認した利用できる SJIS ロケールについてまとめたものです。例えば Red Hat Enterprise Linux の環境において「㈱」のような JIS X 0208 規格にはない拡張文字を扱う場合は、
localedef -f WINDOWS-31J -i ja_JP ja_JP.SJIS
のようにして拡張文字を含んだ charmap ファイルを使ってコンパイルし、ロケールをインストールします。

尚、Red Hat Enterprise Linux[10]をはじめとした Linux OS では SJIS ロケール配下におけるアプリケーションの運用をサポートしていない OS もあるようです。そこで Visual COBOL では SJIS 環境配下で現行運用されるアプリケーションの移行性を高めるべく cobutf8 という Red Hat Enterprise Linux がサポートする UTF8 ロケール配下で SJIS 資産を運用する機能をバージョン 3.0 より実装しました。

9 実際に扱える文字コードはそのプラットフォームでサポートされるロケールに依存します。Visual COBOL との動作がサポートされたロケールについては、AMC ソフトウェアジャパン合同会社の Web ページ中で公開する稼働環境一覧等をご参照ください。

10 【 参考】How to configure to use the ja_JP.SJIS for Japanese locale
https://access.redhat.com/ja/solutions/1218253 (リンク確認: 2024 年 9 月 10 日、登録ユーザー限定のページとなります)

表 1 主な OS の SJIS ロケール／文字コード表

OS	バージョン	ロケール名	コメント
Windows	7,8,10,11 Server 2008 Server 2012 Server 2016 Server 2019 Server 2022 Server 2025	日本語(日本)	Microsoft 社拡張の SJIS(cp932)
Linux	RHEL 6.x RHEL 7.x RHEL 8.x RHEL 9.x Oracle Linux 7.x, 8.x, 9.x	ja_JP.SJIS	JIS X 0208 規格の文字コード charmap ファイル SHIFT_JIS.gz からコンパイルした場合
		ja_JP.SJIS	Microsoft 社拡張の SJIS(cp932) charmap ファイル WINDOWS-31J.gz からコンパイルした場合
		ja_JP.SJIS	JIS X 0213(JIS 拡張漢字) 規格の文字コード charmap ファイル SHIFT_JISX0213.gz からコンパイルした場合
AIX	7.x	Ja_JP.IBM-932	以下の４セットより構成される文字コード - [JISCII] JISX0201 グラフィック左文字セット - [JISX0201] カタカナ/ひらがなグラフィック右文字セット - [JISX0208] 漢字レベル 1 および 2 文字セット - [IBM-udcJP]IBM ユーザー定義可能文字
		Ja_JP Ja_JP.IBM-943	以下の４セットより構成される文字コード - [JISCII] JISX0201 グラフィック左文字セット - [JISX0201] カタカナ/ひらがなグラフィック右文字セット - [JISX0208] 漢字レベル 1 および 2 文字セット - [IBM-udcJP] IBM ユーザー定義可能文字と、NEC の IBM 選択文字 および NEC 選択文字
HP-UX	11.31	ja_JP.SJIS	Microsoft 社拡張の SJIS(cp932) と等価
Solaris	10, 11	ja_JP.SJIS	JIS X 0208 規格の文字コード

4. アーキテクチャーの変更を伴う移行

4.1. 32-bit/64-bit の相違

Visual COBOL は 32-bit アプリケーション、64-bit アプリケーション両者の開発機能を備えます。コンパイルコマンド並びに各ユーティリティもそれぞれのサイズに応じたものが用意されており、適切なワーキングモードを Visual Studio/Eclipse IDE 上で指定することで設定に応じたアーキテクチャー向けのアプリケーションを構築することができます。Linux/UNIX 版の Development Hub であれば cobmode コマンド或いは COBMODE 環境変数を使ってワーキングモードを指定することが可能です。これにより 32-bit アプリケーションを 64-bit アプリケーションへ単純なリコンパイルで格上げすることが可能です。64-bit アプリケーションは拡大された空間を享受できますため、この昇格によりパフォーマンスが向上する可能性があります。例えば、32-bit アプリケーションでは 9 桁までの COMP-5 の数値項目で高い最適化が図れますが、 64-bit 環境ではこれが 18 桁までに拡がります。
ただし、現行の 32-bit アプリケーションが以下のような場合は注意が必要です。

> USAGE POINTER の変数が定義されたプログラム
> これらの変数は 32-bit アプリケーションでは 4 バイト境界で整列されている必要がありますが、64-bit アプリケーションでは 8 バイトの境界で整列されている必要があります。
> USAGE POINTER の変数の使用に整合性がとれていない場合、実行時にメモリ保護違反が起こり得ます。Visual COBOL に付属の Scan64 ユーティリティを利用することで、このようなコードを予め検出できる可能性があります。

> ファイル制御記述（FCD – File Control Description） の利用
> FCD はハンドリング中のファイルの情報が格納されるエリアです。これには 32-bit 用に FCD2、64-bit 用に FCD3 があり、それぞれ xfhfcd2.cpy、xfhfcd3.cpy を展開して利用します。プログラム中で直接 xfhfcd2.cpy を取り込んでいる場合は修正が必要となります。xfhfcd.cpy を読み込んでいる場合は、ワーキングモードの変更によりモードに応じた適切なコピーファイルが取り込まれます。

> 64-bit モードでサポートされないコンパイラ指令の指定
- FASTLINK
- FIXOPT
- SCHEDULER

4.2. ネイティブからマネージへの移行

Visual COBOL は COBOL プログラムから .NET の IL コードや Java bytecode を生成させる機能を備えます。この機能を活用することで既存のネイティブ COBOL プログラムを書き換えることなく .NET のクラスや Java のクラスとして利用することができます。ただし、以下についてはこのマネージ COBOL 機能ではサポートされないため、移行にあたり注意が必要です。これらのうちの大半は .NET や Java の世界での継続利用が非現実的なものとなります。更に、Visual COBOL は .NET や Java の世界と親和性をもたせたモジュールを生成させるための文法やコンパイラ指令を用意しており、段階的にこれらを取り入れていくことでメンテナンス性を一層向上させることが可能です。

> COBOL 文法上の制限
- SCREEN 節
- ACUCOBOL-GT 互換向けに提供される拡張 ACCEPT 文及び DISPLAY 文
- ALTER 文
- CHAIN 文
- XML PARSE 文（JVM COBOL のみ未サポート）
- ネスト化したプログラム中における GLOBAL 句
- メインフレームポインター

> サポートされないライブラリルーチン(※J: JVM COBOL でのみ未サポート)

- CBL_CFGREAD_DYNFH
- CBL_CFGREAD_EXTFH
- CBL_CLEAR_SCR
- CBL_DEBUGBREAK ※J
- CBL_DEBUG_START
- CBL_DEBUG_STOP
- CBL_FREE_RECORD_LOCK
- CBL_GET_CSR_POS
- CBL_GET_KBD_STATUS ※J
- CBL_GET_MOUSE_MASK
- CBL_GET_MOUSE_POSITION
- CBL_GET_MOUSE_STATUS
- CBL_GET_PROGRAM_INFO function 8, 10
- CBL_GET_RECORD_LOCK ※J
- CBL_GET_SHMEM_PTR ※J
- CBL_GET_SCR_GRAPHICS
- CBL_GET_SCR_LINE_DRAW
- CBL_GET_SCR_SIZE
- CBL_HIDE_MOUSE
- CBL_INIT_MOUSE
- CBL_MEM_STRATEGY
- CBL_MEM_VALIDATE
- CBL_READ_KBD_CHAR ※J
- CBL_READ_MOUSE_EVENT
- CBL_READ_SCR_ATTRS
- CBL_WRITE_SCR_ATTRS
- CBL_WRITE_SCR_CHARS
- CBL_WRITE_SCR_CHARS_ATTR
- CBL_WRITE_SCR_CHATTRS
- CBL_WRITE_SCR_N_ATTR
- CBL_WRITE_SCR_N_CHAR
- CBL_WRITE_SCR_N_CHATTR
- CBL_WRITE_SCR_TTY
- C$NARG
- C$PARAMSIZE
- mFFH
- MFFH_MODIFY_DISABLE
- MFFH_MODIFY_TRACE
- MF_CLIENT_STATE_ALLOCATE
- MF_CLIENT_STATE_DELETE
- MF_CLIENT_STATE_EXPIRY
- MF_CLIENT_STATE_FILE
- MF_CLIENT_STATE_PURGE
- MF_CLIENT_STATE_RESTORE
- MF_CLIENT_STATE_SAVE
- PC_ISAPI_GET_EXT
- PC_READ_DRIVE
- PC_SET_DRIVE
- CBL_READ_SCR_CHARS
- CBL_READ_SCR_CHATTRS
- CBL_SCR_ALLOCATE_COLOR
- CBL_SCR_ALLOCATE_VC_COLOR
- CBL_SCR_CREATE_VC
- CBL_SCR_DESTROY_VC
- CBL_SCR_GET_ATTRIBUTES
- CBL_SCR_GET_ATTR_INFO
- CBL_SCR_NAME_TO_RGB
- CBL_SCR_QUERY_COLORMAP
- CBL_SCR_RESTORE
- CBL_SCR_RESTORE_ATTRIBUTES
- CBL_SCR_SAVE
- CBL_SCR_SAVE_ATTRIBUTES
- CBL_SCR_SET_ATTRIBUTES
- CBL_SCR_SET_PC_ATTRIBUTES
- CBL_SET_CSR_POS
- CBL_SET_MOUSE_MASK
- CBL_SHOW_MOUSE
- CBL_SWAP_SCR_CHATTRS
- CBL_TERM_MOUSE
- CBL_TEST_RECORD_LOCK
- CBL_THREAD_CREATE (bit 4 フラグ) ※J
- CBL_THREAD_CREATE_P (bit 4 フラグ) ※J
- CBL_THREAD_DETACH
- PC_WIN_HANDLE
- X"91" function 11 ※J
- X"91" function 12 ※J
- X"91" function 13 ※J
- X"91" function 14 ※J
- X"91" function 15 ※J
- X"91" function 16
- X"91" function 35 ※J
- X"91" function 46 ※J
- X"91" function 47 ※J
- X"91" function 48 ※J
- X"91" function 49 ※J
- X"91" function 69 ※J
- X"A7" function 6, 7
- X"A7" function 16
- X"A7" function 17
- X"A7" function 20, 21
- X"A7" function 25
- X"AF" function 18
- X"B0" function 0
- X"B0" function 2
- X"B0" function 4
- X"E5" ※J

> サポートされないコンパイラ指令

- ACCEPTREFRESH
- ACTUAL-PARAMS
- BOUNDOPT
- CALL-RECOVERY
- CHECK
- COBIDY
- DATA-CONTEXT
- DB2
- DEFAULTCALLS
- FASTCALL
- FASTINIT
- FASTLINK
- FCDALIGN
- FIXOPT
- FP-ROUNDING
- GNT
- HOSTARITHMETIC
- HOSTCONTZERO
- INITPTR
- INT
- INTLEVEL
- LINKCHECK
- LITLINK
- LNKALIGN
- MAPNAME
- NATIONAL
- NATIVE-FLOATING-POINT
- OBJ
- ODOOSVS
- OPT
- PARAMCOUNTCHECK
- PC1
- PERFORMOPT
- PPLITLINK
- PREPROCESS
- PROTECT-LINKAGE
- RDFPATH
- RECMODE
- RECURSECHECK
- REMAINDER
- REPOSITORY
- SCHEDULER
- SEG
- SIGNDISCARD
- SOURCEASM
- SSRANGE
- STICKY-LINKAGE
- TESTCOVER
- TRICKLE
- TRUNCCALLNAME

> サポートされない組み込み関数

- CHAR-NATIONAL

> サポートされないデータベースアクセス

- COBSQL
- DB2 ECM
- OpenESQL(ODBC を介すネイティブモード)

> その他のサポートされない機能

- ネイティブのオブジェクト指向型プログラム
- C プログラムの CALL
- アセンブラサブプログラムの CALL
- Win32 API ルーチンの CALL
- シグナルハンドラーのポスト
- ランタイムスイッチ
- ANSI デバッグ
- CGI 開発
- ACUCOBOL-GT コンパイラオプション -Dz
- ライブラリファイル .lbr への出力
- LPT 機器への出力
- テストカバレッジ

4.3. ACUCOBOL からの移行

ACUCOBOL-GT は旧 Acucorp 社が独自に開発を進めた COBOL 開発製品です、本製品は、固有の COBOL 方言を使って COBOL アプリケーションを開発することが可能です。この特有な方言や機能を持つ ACUCOBOL-GT を使って開発された COBOL プログラムであっても、移行を可能とするための仕組みが Visual COBOL には備わっています。以下はその互換機能の概要となります。

> DIALECT "ACU" コンパイラ指令の指定
> ACUCOBOL-GT の予約語の解釈、ACUCOBOL-GT の挙動に合わせたデータ処理 等
> ACUOPT コンパイラ指令の指定
> ACUCOBOL-GT にてサポートされる主なコンパイラ指令を解釈し、Visual COBOL で利用可能なモジュールへコンパイル
> Visual COBOL 用にカスタマイズされた ACUCOBOL-GT のコンパイルコマンド ccbl
> ACUCOBOL-GT にて開発していた際に指定していたコンパイルコマンドを使って、Visual COBOL のモジュール形式 .int、.gnt を生成

ただし、下記に示す機能については互換性サポートの対象外となります。

- ACUCOBOL-GT マルチスレッドモデル
- ACUCOBOL-GT シンクライアント
- Graphical Technology (GT)
- ACUCOBOL-GT の構成ファイル及び構成変数
- 画面節における ALTER 句, BEFORE 句, 及び EXCEPTION 句
- Management COBOL 開発における -Dz の Truncation オプション (Native 開発では互換サポート)
- パイプを含んだファイル名の ASSIGN

5. 動作不定となるケース

COBOL 言語の国際規格書では、その付録で動作の規定されないようなケースについてまとめています。ここではそれらのうち COBOL 製品で問題になりうるものについて、非互換の実際と考えうる対策について個別に説明します。
以下に述べる救済策は決して最初から採用するべきものではありません。正しく書かれたアプリケーションを正しいデータで運用する限りこれらの非互換に遭遇することはありません。これらの救済策を取ることによって性能劣化などの副作用が起こりうる点をご留意いただき対策を検討してください。

5.1. 算術式の桁あふれ条件

言語仕様上の規定	ON SIZE ERROR 指定がされず、かつ算術文によって指定された算術演算の実行後桁あふれ条件が起きると、その結果の一意名の値は規定しない。
違反による影響	コンパイラの最適化技術向上で桁あふれが無いという条件のもとに効率的なオブジェクトコード生成がなされる。
救済策	> HOST-ARITHMETIC コンパイラ指令の指定 最適化が弱い旧版製品と同様に桁あふれ分を切り捨て。ただし、該当の算術文は最適化されない。 この指令の狙いはあくまでも IBM メインフレームの COBOL との互換性であって旧版の COBOL 製品からの互換性を保証するものではない。しかし、過去のバージョンアップ事例でこれを使用して非互換を部分的に回避した事例はあり。 > CHECKDIV コンパイラ指令の指定 0 除算を検出するとその時点でその旨の実行時エラーを出力。ただし、該当の算術文は最適化されない。

5.2. 項目定義と矛盾するデータの参照

言語仕様上の規定	字類条件を除いて、手続き部でデータ項目の内容が参照されるとき、データ項目の内容が PICTURE 句によるデータ項目の定義と矛盾する場合には、手続き部での結果は、規定しない。
違反による影響	コンパイラの最適化技術向上で正当なデータが格納されているという条件のもとに効率的なオブジェクトコード生成がなされる。
救済策	> CHECKNUM コンパイラ指令の指定 数値項目には数値のみが格納されるよう実行時にチェックするコードを生成。この分のオーバヘッドは発生する。 > SPZERO コンパイラ指令の指定 USAGE DISPLAY の数値項目にスペースが格納されている場合、本指令を指定すると同値を 0 として扱う。 > SIGN-FIXUP コンパイラ指令の指定 本指令を指定すると不正な符号ビットを IBM メインフレームの COBOL コンパイラの NUMPROC(NOPFD) 指令と同様に修正して計算する。このため旧版の COBOL コンパイラの低い最適化レベルでの結果との互換性が高まる。 この指令も狙いはあくまでも IBM メインフレームの COBOL との互換性であって旧版の COBOL 製品からの互換性を保証するものではない。しかし、過去のバージョンアップ事例でこれを使用して非互換を部分的に回避した事例はあり。 本指令による NUMPROC(NOPFD) の限定的なエミュレート機能を有効にするには HOST-NUMCOMPARE 及び HOST-NUMMOVE 指令も併せて指定する。 > Eclipse IDE/Visual Studio IDE の利用 Visual COBOL をインストールすると Eclipse IDE 及び Visual Studio IDE にて COBOL 用に作りこまれたデバッガを利用可能。このデバッガを用いて条件付きのブレークポイントを指定し問題となりうるコードを検出。

5.3. 項目定義と矛盾するデータの転記

言語仕様上の規定	[MOVE 文] 受取り側の項目が数字または数字編集であり送出し側の項目が英数字である場合、送出し側の項目の内容が整数でないと、結果はどうなるかわからない。
違反による影響	仕様で規定されているように不定の動作。 以下の COBOL 文や句においてもデータの転記が発生し、MOVE 文と同じルールが適用される。従って、これらにおいても上の条件に該当するようなコーディングがされていると動作は規定されない。 - VALUE 句 - ACCEPT 文 - INITIALIZE 文（REPLACING 指定あり） - READ 文（INTO 指定あり） - REWRITE 文（FROM 指定あり） - WRITE 文（FROM 指定あり）
救済策	> SPZERO コンパイラ指令の指定 USAGE DISPLAY の数値項目にスペースが格納されている場合、本指令を指定すると同値を 0 として扱う。 > SIGN-FIXUP コンパイラ指令の指定 本指令を指定すると不正な符号ビットを IBM メインフレームの COBOL コンパイラの NUMPROC(NOPFD) 指令と同様に修正して計算する。このため旧版の COBOL コンパイラの低い最適化レベルでの結果との互換性が高まる。 この指令も狙いはあくまでも IBM メインフレームの COBOL との互換性であって旧版の COBOL 製品からの互換性を保証するものではない。しかし、過去のバージョンアップ事例でこれを使用して非互換を部分的に回避した事例はあり。 本指令による NUMPROC(NOPFD) の限定的なエミュレート機能を有効にするには HOST-NUMCOMPARE 及び HOST-NUMMOVE 指令も併せて指定する。 > 組み込み関数 NUMVAL/NUMVAL-C の利用 COBOL では ANSI 85 規格にて、英数字項目を数値に変換するための組み込み関数 NUMVAL 及び NUMVAL-C が規定済み。これらの関数を利用し、明示的に英数字項目を数値項目へ変換。 > Eclipse IDE/Visual Studio IDE の利用 Visual COBOL をインストールすると Eclipse IDE 及び Visual Studio IDE にて COBOL 用に作りこまれたデバッガを利用可能。このデバッガを用いて条件付きのブレークポイントを指定し問題となりうるコードを検出。

5.4. 転記の受け側と送り側の重なり

言語仕様上の規定	ある文中の送り出し側項目と受取り側項目が同じデータ記述項によって定められたものでない記憶領域の一部又は全部を共有するとき、そのような文の実行結果は、規定しない。
違反による影響	仕様で規定されているように不定の動作。
救済策	コンパイラによる検出 WARNING“3“ を指定した場合、部分参照や添え字を使用していない項目の明示的または暗黙的な MOVE 文についてはコンパイラが作用対象の重なりを検出。

5.5. 適合しない CALL 文パラメーター

言語仕様上の規定	呼び出し元プログラムの CALL 文に指定しているパラメーターの数、サイズ、およびデータ型が、呼び出し対象プログラムの該当パラメーターと一致していること。呼び出し元プログラム内の REFERENCE、CONTENT、および VALUE 指定の用法が、呼び出し対象プログラムのパラメーター定義と一致していること。 COBOL で書かれていないプログラムを呼ぶ CALL 文を使うときの復帰の方法及びプログラム間のデータ連絡については、規定しない。
違反による影響	不定の動作。場合によっては、メモリ保護違反等の実行時エラー。不整合による影響は、一致していないパラメーターでやり取りされるデータや、使用する呼び出し規約によって大幅に異なる。
救済策	> CALL-CONVENTION 句を使って呼び出し規約を指定 特殊名段落に CALL-CONVENTION 句を指定し COBOL – 他言語プログラムの間で呼び出す際の呼び出し規約を指定。 > プログラムのマネージ化 Visual COBOL が備える .NET / JVM COBOL の機能を利用しプログラムをマネージ化。 これにより、コンパイル時点でパラメーターの数、サイズ、データ型等の不一致を検知。

5.6. ソートされていないデータのマージ

言語仕様上の規定	ファイル名 2 及びファイル名 3 のファイル上のレコードが、MERGE 文 ASCENDING KEY 指定または DESCENDING KEY 指定で指定された通りに並んでいなければ、併合操作の結果は規定しない。
違反による影響	旧版の COBOL 製品では何のエラーも出ずに不正なマージ結果となっていたが、 Server Express/Net Express 5.1 以降、及び Visual COBOL では明示的にエラーを報告するように改善された。
救済策	ソートユーティリティを使った事前ソート Visual COBOL に付属するソートユーティリティ MFSORT を使用し、プログラム実行前に対象のファイルをソート。

5.7. 初期化されていないデータ項目

言語仕様上の規定	作業場所節又は通信節のデータ項目に VALUE 句を書かなければ、そのデータ項目の初期値は、規定しない。
違反による影響	DEFAULTBYTE コンパイラ指令の指定内容に従属。
救済策	> DEFAULTBYTE コンパイラ指令の指定 VALUE 句が指定されていない項目を DEFAULTBYTE コンパイラ指令に指定した値で初期化する。 COBOL 製品では一貫して DEFAULTBYTE のデフォルトのパラメーターを 32 (16 進で言えば X'20'、つまり空白値) としている。本指令を明示的に指定していない限り空白で初期化されており、非互換はない。 > INIT-BY-TYPE コンパイラ指令の指定[11] VALUE 句の指定がなく且つ REDEFINES 句や EXTERNAL 属性のない WORKING-STORAGE SECTION 中の項目を下記のルールに基づき初期化する。 - 英字、英数字、英数字編集、数値編集項目 ・・・ 空白 - 数値項目 ・・・ 0 - ポインター ・・・ null - インデックス項目 ・・・ 1

11 Visual COBOL 2.2 Update 1 より実装された機能となります。

5.8. ファイル節の VALUE 句

言語仕様上の規定	ファイル節のデータ項目の初期値は規定しない。INITIALIZE 文が発行されない限り初期値はセットされない。
違反による影響	Server Express/Net Express 4.0 以前では VALUE 句にてセットした値が有効となったケースがあり。以降のバージョン / 製品ではコンパイラの改善により、言語規約に則りこの処理を撤廃し最適化を実現。
救済策	> INITIALIZE 文を使ってデータ項目を初期化 > VALUE 句を持つファイル節中のデータ項目を作業場所節に転記 新たに作業場所節に VALUE 句を持つデータ項目を用意し、そこからファイル節中のデータ項目へデータ転記。

5.9. 読み込むデータのレコード長がレコード記述項で指定したレコード定義より小さい

言語仕様上の規定	現在のレコード領域の範囲を超えてデータ項目が存在する場合、READ 文の実行が終了したときにその内容はどうなっているかわからない。
違反による影響	言語仕様上で規定されているように読み込んだレコードの長さを超えた位置に存在するデータ項目への格納値は規定されない。
救済策	ファイルハンドラー構成オプション SPACEFILL 行順ファイルであれば、SPACEFILL を ON(デフォルト値) にして空白文字で埋めるよう構成。

5.10. 複数の PERFORM 文でその実行対象範囲に重なりがある

言語仕様上の規定	PERFORM 文をネストすることは可能であるが、ネストされた PERFORM 文の実行範囲は親の実行範囲の完全内部にあるか完全外部であること。またネストされた PERFORM 文は親と EXIT を共有はできない。
違反による影響	下記のように実行対象範囲に重なりがあり尚且つ EXIT を共有するようなプログラムの動作は規定されない。 PROCEDURE DIVISION. MAIN-PROC. PERFORM A THRU D STOP RUN. A. PERFORM B THRU D B. ・・・ C. ・・・ D. ・・・
救済策	コンパイラ指令 PERFORM-TYPE コンパイラ指令 PERFORM-TYPE に COBOL370, ENTCOBOL, OS390, OSVS, VSC2 のいずれかのパラメーターを指定することで 2 階層までであれば実行対象範囲の重なりを許可。

5.11. GO TO 文などによって明示的に終了しない PERFORM 文

言語仕様上の規定	PERFORM 文による手続き実行はいくつかの論理経路があってもよいが、その実行範囲で終了する必要がある。つまり、GO TO 文により外部へ分岐することは可能だが、必ず再び PERFORM 文の実行範囲に戻って終了する必要がある。
違反による影響	終了しないまま処理を続行していると COBOL ランタイムはまだ PERFORM 文を実行中の状態で処理を行うためプログラマが想定した挙動を得られない可能性がある。
救済策	違反ロジックの修正 Visual COBOL に装備されたコード分析機能には単純な条件であればこのような違反を検出するクエリが用意されています。 より複雑な条件で違反を検出する場合は Enterprise Analyzer という製品を活用して要件に合ったクエリを用いることができます。本製品には様々な条件で改善候補とすべきロジックを検出するためのクエリがビルドインされているだけでなく、要件に応じてクエリをカスタマイズ作成する機能も準備されています。 これらのクエリは Visual COBOL のコード分析機能に取り込んで利用することも可能です。 このようにして問題となり得るコードを検出して対策を講じます。

5.12. 範囲外の添え字付け

言語仕様上の規定	OCCURS 句を伴って定義された表の参照時に範囲外の添え字が指定されると、コンパイラやランタイムがこれを検知しエラーを返す。この動作はコンパイラ指令 NOBOUND を指定すると無効化され、この状態で範囲外の添え字を伴う参照等を行った場合の動作結果は規定されない。
違反による影響	下記のように範囲外の添え字が指定された参照や転記の結果は規定されない。 WORKING-STORAGE SECTION. 01 WK-9 PIC 9(1). 01 WK-TBL. 03 WK-X PIC X(1) OCCURS 5. PROCEDURE DIVISION. MOVE 6 TO WK-9. MOVE 'Z' TO WK-X(WK-9).
救済策	コンパイラやランタイムによる検出 既に指定されているコンパイラ指令 NOBOUND または NOCHECK を外すか、あるいはコンパイラ指令 BOUND または CHECK を指定することで、コンパイル時や実行時に指定された添字が OCCURS 句で定義された範囲内にあるかチェックされる。

5.13. 範囲外領域の部分参照

言語仕様上の規定	部分参照において、左端の位置および長さの評価が有効な値を生成することはチェックされない。これらの評価がこの規則で述べられた制限に適合しない場合、結果は未定義になり、他のデータ項目は破損する可能性がある。
違反による影響	下記のように範囲外の領域が指定された部分参照の動作は規定されない。 WORKING-STORAGE SECTION. 01 WK-9 PIC 9(1). 01 WK-X PIC X(5) VALUE 'ABCDE'. PROCEDURE DIVISION. MOVE 6 TO WK-9. MOVE 'Z' TO WK-X(WK-9:1).
救済策	コンパイラやランタイムによる検出 コンパイラ指令 SSRANGE を指定することで、コンパイル時や実行時に部分参照、添え字、索引の境界がチェックされる。

6. 付表

6.1. COBOL コンパイラの改善・改修により顕在化された主な事象

No.	機能概要	コード例
1	Visual COBOL のデフォルト指令 MF(18) は DBSPACE 指令を有効にします。そのため、DBSPACE を無効にする関連の指令もしくは DBSPACE 指令そのものを明示的に無効にしない限り、デフォルトで本指令は有効です。Visual COBOL の 1 世代前の製品 Net Express 及び Server Express に関しても MF(7) 以上が指定されるためこの点は同様です。しかし、NONCHAR 指令を指定すると DBSPACE 指令がコンパイラに渡っていても INITIALIZE 文に関してはそれが効いていないかのような挙動となる障害が過去バージョンでは潜在しており、2.2 Update 1 で改修されました。	WORKING-STORAGE SECTION. 01 TEST01. 03 TEST01-N PIC N(10). PROCEDURE DIVISION. INITIALIZE TEST01. * 上記オペレーションで全角スペースが格納されないことがありました。
2	下記環境では、バイナリ及びパック十進形式の数値項目を DISPLAY 文で出力する場合、ゾーン十進項目に変換して出力します。しかし、負の数の場合は最下桁にオーバーパンチ文字を出力します 。そのため、これらの互換性機能を有効にすると互換元の環境に合わせてオーバーパンチ文字を出力させますが、旧製品のあるバージョンまでは、互換性機能を有効にしても国際規格に合わせて、符号を独立させたゾーン十進項目として出力させていました。 > IBM COBOL/370 > IBM COBOL for OS/390 > IBM COBOL for MVS > IBM DOS/VS COBOL > IBM Enterprise COBOL for z/OS > IBM OS/VS COBOL > IBM VS COBOL II Visual COBOL 3.0 からは、USAGE DISPLAY で数字項目を出力する場合も同様の事象が発生します。	$SET ENTCOBOL DATA DIVISION. WORKING-STORAGE SECTION. 01 AITEM PIC S9(4) COMP-3 VALUE -1234. PROCEDURE DIVISION. DISPLAY AITEM. * この場合「123t」のようにして出力されます。 * しかし、あるバージョンまでは「-1234」のように出力していました。 GOBACK.
3	マネージ COBOL で PERFORM で呼ばれた SECTION 内の EXIT METHOD が正常に動作しない不具合が過去バージョンに潜在しており、Visual COBOL 4.0 で改修されました。	METHOD-ID. TEST01 IS PUBLIC. WORKING-STORAGE SECTION. PROCEDURE DIVISION. PERFORM SEC01. DISPLAY "あいうえお". * 旧バージョンでは SEC01 内の EXIT METHOD が正常に動作せず、 * PERFORM 後の当該ステップが実行されていました。 GOBACK. SEC01 SECTION. EXIT METHOD. END METHOD TEST01.

No.	機能概要	コード例
4	Visual COBOL 3.0 より、SCROLLOPTION=STATIC 指定した静的カーソルを利用した検索処理は、SCROLLOPTION=DYNAMIC 指定の動的カーソル利用時と比べ、検索性能が劣化する場合があります。事象の回避策として、動的カーソルの利用をお願いします。 本不具合は 5.0 にて改修されました。	EXEC SQL SET SCROLLOPTION STATIC END-EXEC. *STATIC 指定のため、検索性能が劣化する場合があります。 EXEC SQL OPEN myCursor END-EXEC. PERFORM UNTIL SQLSTATE = '02000' EXEC SQL FETCH CURDBFETCH INTO :HOSTVAL END-EXEC
5	DBCHECK 指令による、日本語定数内に設定された半角スペース (x'20') の扱いを正しく検知できない不具合が過去バージョンに潜在しており、Visual COBOL 3.0 にて改修されました。NODBCHECK 指令により、旧バージョンと同様に動作させることができます。	01 nval1 PIC N(02) VALUE N" あ". *" あ" の前に半角スペースが 1 つだけ設定されており、 *これは DBCHECK によりエラーとして報告されるべきでしたが、 *旧バージョンでは報告されませんでした。
6	Net Express では、SQL コンパイラ指令 : ALLOWNULLCHAR を指定することなく、SQL Server の CHAR 型のカラムに対し、PIC X(n) のホスト変数を利用したバイナリデータの選択、挿入、更新が行えました。 Visual COBOL では、SQL コンパイラ指令 : ALLOWNULLCHAR 指定がないと、バイナリデータを正しく処理できません。本来は、どちらも ALLOWNULLCHAR を指定するのが正しい利用方法です。	DB-VAL1 PIC X(02). … EXEC SQL INSERT INTO tempDB VALUES (:DB-VAL1) *DB-VAL1 にバイナリデータが設定された場合、 *ALLOWNULLCHAR 指定無しでは *DB 登録時に文字化けします。 END EXEC.
7	DBCS 型の PIC N 項目の宣言時に、各国語型の値を転記することはできません。しかし、Net Express, Visual COBOL 4.0 までは、ALL 指定を行った際にエラーとならない不具合がありました。5.0 では、正しくコンパイルエラーになります。	WORKING-STORAGE SECTION. 01 A PIC N VALUE NX'3000'. 01 B PIC N VALUE ALL NX'3000'. *上記 ALL 指定無しの行と同様、エラーとして報告されるべきでしたが、 *旧バージョンでは報告されませんでした。 PROCEDURE DIVISION.
8	Server Express 5.1 WrapPack 11 以前、Visual COBOL 2.2 Update 1 以前では、コンパイラ指令 CHECKNUM を指定して生成した GNT/EXE が、空白を数字項目に代入する文で実行時エラー 163 が発生しない不具合があります。 この不具合は、Server Express 5.1 WrapPack 12、Visual COBOL 2.2 Update 2 にて改修されています。	WORKING-STORAGE SECTION. 01 XVAL PIC X(02). 01 NVAL PIC 9(02). PROCEDURE DIVISION. MAIN-PROC SECTION. MOVE "AA" TO XVAL. MOVE XVAL TO NVAL. DISPLAY NVAL UPON CONSOLE. * 上記で実行時エラー 163 が発生すべきでしたが、 * 以前のバージョンではエラーとならず、処理が継続されました。
9	PIC N 項目に対して NUMERIC の比較は言語仕様上サポートされておらず、エラーとして報告すべきでしたが、3.0 Patch Update 3 まではエラー報告できない不具合がありました。 3.0 Patch Update 4 以降では、"114 メモリ領域外の項目にアクセスしようとしている (シグナル 11)" のランタイムエラーが発生します。5.0 Patch Update 7 にて、"303-S 作用対象が誤ったデータ形式である" エラーとして報告されるように改修されました。	WORKING-STORAGE SECTION. 01 NVAL PIC N(1). PROCEDURE DIVISION. IF NVAL IS NUMERIC * 上記でエラーが報告されるべきでしたが、旧バージョンでは * 報告されませんでした。 END-IF. GOBACK.

No.	機能概要	コード例
10	COBOL 言語仕様上 COMPUTE 文に ROUND 句がない場合は切り捨てとなるべきですが、 .NET COBOL と JVM COBOL で Visual COBOL 2.1 以前では切り捨てとならない不具合があります。 この不具合は、Visual COBOL 2.2 にて改修されました。	WORKING-STORAGE SECTION. 01 A PIC S9(11) VALUE 20. 01 B PIC S9(11) . 01 C PIC S9(3)V9(8). PROCEDURE DIVISION. COMPUTE B = ((A / 3) - 107) * 3. DISPLAY B. COMPUTE C = ((A / 3) - 107) * 3. DISPLAY C. 【ネイティブ COBOL の結果】 00000000300- 30099999999- 【.NET COBOL / JVM COBOL の結果】 00000000301- 30100000000-
11	NATIONAL 項目と集団項目の比較は、NATIONAL 項目と集団項目との転記の規則に準拠し、英数字比較が行われます。しかし、Visual COBOL 4.0 より前の製品では、正しくこの比較が行えませんでした。	$SET NSYMBOL(NATIONAL) PROGRAM-ID. PROGRAM1 AS "PROGRAM1". DATA DIVISION. WORKING-STORAGE SECTION. 01 IN-VAR. 03 IN-VAR1 PIC N(15). 03 IN-VAR2 PIC N(05). 01 VAR PIC N(20). PROCEDURE DIVISION. MOVE ALL N"あ" TO IN-VAR1. MOVE ALL N"あ" TO IN-VAR2. MOVE ALL N"あ" TO VAR. IF IN-VAR = VAR * Visual COBOL 4.0 より前の製品では 上記判定は FALSE となります。 DISPLAY "MATCH" UPON SYSOUT ELSE DISPLAY "UNMATCH" UPON SYSOUT END-IF.
12	.NET および JVM COBOL において、INDD または OUTDD コンパイラ指令を使用して生成した実行可能ファイルを実行時に、SYSIN または SYSOUT 環境変数が設定されていない、または指定されたファイルが存在しないにもかかわらず、ACCEPT や DISPLAY 文でエラーが発生しない不具合が改修され、適切なエラーメッセージが表示されるようになりました。過去のバージョンではエラーは発生せず許容されます。 【改修が反映されたバージョン】 VC/ED 4.0 PU18 以降 VC/ED 5.0 PU7 以降 VC/ED 6.0 以降	PROGRAM-ID. PROGRAM1 AS "SAMPLE.PROGRAM1". DATA DIVISION. WORKING-STORAGE SECTION. 01 VAL PIC X(10). PROCEDURE DIVISION. ACCEPT VAL FROM SYSIN. * 以前のバージョンでは、SYSIN 環境変数が未定義であっても * エラーが報告されませんでした。 GOBACK. END PROGRAM PROGRAM1.

No.	機能概要	コード例
13	OpenESQL プリプロセッサを用いた FETCH もしくは SELECT INTO 句を利用したホスト変数へのデータの取得処理において、ホスト変数を集団項目として定義を行い、かつ、TIMESTAMP-RECORD を基本項目以外で利用していた場合に、正しくホスト変数を展開できず、コンパイルエラーが発生していました。これは、SQL(GEN-HV-FROM-GROUP) 指令でも回避できませんでした。 この不具合は、Visual COBOL 6.0 Patch Update 11、7.0 Patch Update 1 にて改修されました。	HOSTVAR.CPY 01 HOSTVAR. 03 DID PIC 9(8). 03 DTIMESTAMP SQL TYPE IS TIMESTAMP-RECORD. PROGRAM1.CBL EXEC SQL BEGIN DECLARE SECTION END-EXEC. COPY HOSTVAR.CPY. EXEC SQL END DECLARE SECTION END-EXEC. … EXEC SQL SELECT ID,TS INTO :HOSTVAR * TIMESTAMP がホスト変数内の集団項目に使用されているため、 * コンパイルエラーになります。 FROM DTABLE WHERE ID=1 END-EXEC. EXEC SQL FETCH CURDTABLE INTO :HOSTVAR END-EXEC.
14	以下の条件に全て合致する場合に数値比較が正しく行えない不具合があり、Visual COBOL 6.0 Patch Update 17, Visual COBOL 7.0 Patch Update 7 にて改修されました。 • Intel CPU の 32-Bit モードでコンパイルしている • HOST-NUMCOMPARE 指令が指定されている • NATIVE(EBCDIC)指令が指定されている、または PROGRAM COLLATING SEQUENCE 句が指定されている • 符号なしの COMP-3 または DISPLAY 項目同士の比較 • 比較の両辺の桁数が一致している • .gnt や .o などの生成コードで実行	WORKING-STORAGE SECTION. 01 T2MAX PIC 9(03) COMP-3 VALUE 3. 01 T2SB PIC 9(03) COMP-3 VALUE 0. PROCEDURE DIVISION. MOVE 5 TO T2SB. IF T2SB > T2MAX *正しくは DISPLAY TRUE 句が実行されるべきですが、 *DISPLAY FALSE 句が実行されました。 DISPLAY 'TRUE' ELSE DISPLAY 'FALSE' END-IF.
15	OpenESQL プリプロセッサを用いた埋め込み SQL で、WHERE 句内にホスト変数を用いた条件を設定する場合、ホスト変数として集団項目を使用することはできません。使用している場合は、プログラムの修正が必要です。	01 KEYBLOCK. 03 KEY-AREA. 05 KEY01 PIC X(08). 01 BOOKINFO. … EXEC SQL SELECT TITLE INTO :DTITLE FROM DTABLE WHERE STOCK_NO = :KEY-AREA * 以前のバージョンではエラーが報告されませんでした。 END-EXEC.

No.	機能概要	コード例
16	OpenESQL プリプロセッサを用いた埋め込み SQL が記述されているプログラムに CASE 式を含む SELECT 文が記述されている場合は、コンパイル時に ALLOWSERVERSELECT SQL 指令オプションを指定する必要があります。未指定の場合、コンパイルエラーとなります。	PROCEDURE DIVISION. EXEC SQL SELECT CASE COL1 WHEN 0 THEN 10 WHEN 1 THEN 100 ELSE 1 END COL1 FROM TBL END-EXEC.
17	RECORD 句にて可変長レコードの文字位置の最小数および最大数を指定する場合は、最大数は最小数より大きい値を指定する必要があります。しかし、Visual COBOL 3.0 までは正しくエラーとして報告できていませんでした。	FILE-CONTROL. SELECT INFILE ASSIGN TO "foo" ORGANIZATION IS SEQUENTIAL. FD INFILE RECORD IS VARYING IN SIZE 10 TO 10 CHARACTERS. *以前のバージョンではエラーが報告されませんでした。
18	マネージド COBOL において、CHECKNUM 指令を指定してコンパイルされたプログラムの実行時に数字項目から数字編集データ項目に MOVE する場合に、送り元に空白文字などの非数値データが含まれているにもかかわらず、エラーが報告されない不具合が過去バージョンに潜在しており、Visual COBOL 6.0 で改修されました。	WORKING-STORAGE SECTION. 01 VAL_Z9 PIC ZZ9. 01 VAL_X PIC X(3). 01 VAL_XR REDEFINES VAL_X PIC 9(3). PROCEDURE DIVISION. MOVE " 30" TO VAL_X. MOVE VAL_XR TO VAL_Z9. * 以前のバージョンではエラーが報告されませんでした。
19	内部 SORT または MERGE 文において、入力レコードの最大サイズが SD に定義されたレコードサイズより大きい、または出力レコードの最大サイズが SD に定義されたレコードサイズより小さい場合にコンパイルエラーにならない不具合が過去バージョンに潜在しており、Visual COBOL 4.0 で改修されました。	FILE-CONTROL. SELECT INFILE ASSIGN TO 'IN.DAT'. SELECT OTFILE ASSIGN TO 'OUT.DAT'. SELECT STFILE ASSIGN TO SYS001. FILE SECTION. FD INFILE. 01 INREC PIC X(80). FD OTFILE. 01 OTREC PIC X(40). SD STFILE. 01 STREC PIC X(40). PROCEDURE DIVISION. SORT STFILE ASCENDING STREC USING INFILE GIVING OTFILE.
20	COBOL 言語の仕様では PIC N 項目の部分参照は各国語文字の文字数でカウントすべきですが、Visual COBOL 3.0 以前のバージョンまでては バイト数単位でカウントされて不具合があり、Visual COBOL 3.0 で改修されました。なお、この問題は PIC N 項目の部分参照付けが受け側となっている転記で、その送り側が集団項目である場合にのみ発生していました。。	01 A PIC N(50). 01 B. 02 PIC X(2) VALUE "A". 02 PIC X(6) VALUE "B C D". PROCEDURE DIVISION. MOVE B TO A(24:4).

No.	機能概要	コード例
21	コンパイラ指令 SIGNFIXUP を伴うプログラムで符号なしの COMP-6 形式の項目間の転記処理が正しく行えない不具合があり、Visual COBOL 8.0 Patch Update 16, 9.0 Patch Update 7 にて改修されました。	$SET SIGNFIXUP WORKING-STORAGE SECTION. 01 A PIC 9(8) COMP-6. 01 B PIC 9(8) COMP-6 VALUE 20240312. PROCEDURE DIVISION. DISPLAY "B = " B ", B(Hex)= " FUNCTION HEX-OF(B). MOVE B TO A. DISPLAY "A = " A ", A(Hex)= " FUNCTION HEX-OF(A).
22	データ項目が明示的にまたは暗黙的に USAGE DISPLAY として記述され、項類が数値、外部浮動小数点数、数字編集、英字、または英数字編集のいずれかであるとき、部分参照の目的上、そのデータ項目は同じ大きさの英数字データ項目として再定義されたものとして操作されます。しかし、英数字データ項目ではなく数値データ項目として扱われる不具合があり、10.0 にて改修されました。本例では、H5-VAR9(1:1) に改修によって半角スペースがセットされるようになりましたが、旧バージョンでは 0 が設定されました。	01 HEAD5. 03 H5-VAR9 PIC 9(03). 03 H5-VARX PIC X(01). PROCEDURE DIVISION. MOVE 111 TO H5-VAR9. INITIALIZE H5-VAR9(1:1) MOVE 'X' TO H5-VARX. DISPLAY HEAD5.
23	NATIONAL 項類から集団項目への転記規則は英数字転記となります。送り側より受け側の項目長が大きい場合にパディングが行われますが、このパディング値は英数字項目扱いとなるため、半角空白となります。しかし、4.0 より前の製品にて、半角空白が設定されない不具合がありました。 本事象により、3.0 で右記のコードを実行した場合は、OK が出力されません。	$SET NSYMBOL(NATIONAL) WORKING-STORAGE SECTION. 01 A PIC N(5) USAGE NATIONAL VALUE N'あいうえお'. 01 B. 05 C PIC X(12). PROCEDURE DIVISION. MOVE A TO B. IF C(11:2) = SPACE DISPLAY 'OK' END-IF.

6.2. コンパイラやランタイムの機能拡張に伴い優先される機能の変更

No.	コンポーネント	変更概要	変更による主な影響／対策例
1	コンパイラ	Visual COBOL では中間コードの生成技術についても改善しています。これにより、旧製品をベースとしたレベルでの中間コードは生成されません。それに伴い、Visual COBOL では、INTLEVEL コンパイラ指令を指定する必要がなくなりました 。	Visual COBOL 製品では、コンパイル時に INTLEVEL を明示指定した場合も無視され、プログラム動作に一切影響を及ぼしません。将来のメンテナンス時における混乱を避けるためにも、指定の削除を推奨します。
2	コンパイラ	再帰呼び出し検出機能を有効にするコンパイラ指令 RECURSECHECK はデフォルトでは無効でしたが、デバッグモードでコンパイルされた場合は有効となります。ただし、REENTRANT 指令が指定された場合は RECOURCECHECK は有効化されません。	RECOURSECHECK 指令が有効になっている場合、LOCAL-STORAGE SECTION を持たないプログラムの再帰呼び出しされると実行時エラーが戻されます。 このエラーを回避するためには、NORECURSECHECK 指令を設定してください。
3	ファイルハンドラー	Visual COBOL よりファイルハンドラー構成オプション FILEMAXSIZE のデフォルト値が 8 となり、大容量ファイルの扱いがデフォルトで可能となりました。(Net Express、Server Express ではデフォルト値は 4 となります。)	一つのファイルを FILEMAXSIZE=4 で稼働するプログラムと FILEMAXSIZE=8 で稼働するプログラムとが同時に開き更新を行うと、ファイルが破損する可能性があります。 FILEMAXSIZE の設定値が不一致となるプログラムが混在しないよう設定・運用を行ってください。 同一製品で作成したファイルであっても起こり得ますが、特にデフォルト値が異なる製品移行時には注意が必要です。
4	ファイルハンドラー	Visual COBOL はコンパイラ指令 IDXFORMAT 並びにファイルハンドラー構成オプション IDXFORMAT のデフォルト値の解釈を下記のように変更しました。 【Visual COBOL】 デフォルト値: 0 ・ 8(大容量索引ファイル)と同等 【Server Express 5.1】 デフォルト値: 0 ・ 1(C-ISAM 形式)と同等 【Net Express 5.1】 デフォルト値: 0 ・ 3(Micro Focus 索引形式)と同等	Visual COBOL より論理的なファイルサイズの上限が無制限で且つ高速な処理が可能な索引ファイル形式のフォーマットがデフォルトとなりました。このフォーマットは旧製品でデフォルトとして指定されていた索引部を .idx の拡張子を持つ別ファイルに切り出すフォーマットと異なり、索引部もファイル中に含めます。そのため、旧環境で .idx も含めたファイルコピー等が運用手順に含まれていたのであれば不要となります。 また、Visual COBOL 2.3 Update 1 より索引部を .idx の別ファイルに切り出し尚且つフォーマット「8」のようにファイルサイズの上限が無制限で且つ高速処理が可能な新フォーマット「12」が実装されました。本フォーマットを指定すればフォーマット「8」の機能を享受しつつ .idx を意識した運用を維持できます。
5	OpenESQL	OpenESQL 指令 CHECKSINGLETON のデフォルト値が Visual COBOL 2.2 Update 1 より「NOCHECKSINGLETON」から指定なしに変更されました。	複数行が返るようなクエリーを SELECT INTO 文で発行し且つ CHECKSINGLETON を明示指定していない場合、Visual COBOL 2.2 以下では正常処理された旨の SQLCODE=0 を返していました。一方、Visual COBOL 2.2 Update 1 以降におけるデフォルト動作では +1 を SQLCODE に格納し注意喚起します。旧バージョン同様に複数行が返ってきても SQLCODE に 0 を格納させたい場合は明示的に NOCHECKSINGLETON を指定します。
6	OpenESQL	.NET COBOL 開発の際 OpenESQL 指令 DBMAN に「ODBC」を指定するには .NET Framework のターゲットを 4.0 以上とする必要があります。	旧環境に合わせて .NET Framework のターゲットを 3.5 以下に落とすとコンパイルが通りません。この場合は .NET Framework 4.0 以上をターゲットとする、或いは ADO.NET による Database 接続を検討します。
7	OpenESQL	AS 句で接続名を明示指定しない CONNECT 文による接続中に、再度 AS 句で接続名を明示指定しない CONNECT 文を実行すると、「重複した接続」のエラーとなる。	OpenESQL の仕様としては、同時に複数の接続を使用する場合には AS 句で明示的に接続名を指定する必要があります。Visual COBOL 3.0 以前ではこれについての実行時チェックが曖昧で、デフォルト接続を上書きするような動作になっていました。 正しい使用方法に準拠して接続名を明示指定するように変更する必要があります。
8	DB2 ECM	DB2(BIND) 指令によるコンパイル時に生成されるバインドファイル (.bnd) 中で、プログラム中に書かれたカーソル名に含まれるハイフン '-' がアンダースコア '_' に変換されます。 通常の SQL 文の実行では問題ありませんが EXECUTE IMMEDIATE 文によってカーソル名を含む SQL 文を動的に実行する場合に、旧バージョンでは発生していなかったエラーが発生します	メインフレーム DB2 との互換性を向上させる HCO (Host Compatibility Option) 機能がデフォルトで有効になったことにより、DB2 LUW では許容されていないハイフンを回避するような機能追加がなされました。 DB2(NOHCO) 指令を明示的に指定して再コンパイルすることによって回避することができます。

No.	コンポーネント	変更概要	変更による主な影響／対策例
9	DB2 ECM	DB2 指令を LITLINK 指令と併用して実行形式にリンクすると、シンボル未解決エラーとなる。	HCO 機能がデフォルトで有効になったことにより、この機能を実装する動的ロード形式のランタイムモジュールが内部的に呼び出されているために発生しました。 DB2(NOHCO) 指令を明示的に指定して再コンパイルすることによって回避することができます。なお、Windows では NOPRE オプションも合わせて指定ください。
10	コンパイラ	Visual COBOL のコンパイラは様々な文字集合で書かれた COBOL ソースに一律で対応するために、読み込んだソースコード行をいったん Unicode に変換してから構文解析を行っています。このため、ソースコード中の文字定数のエンコーディングと合致したロケールで環境変数：LANG を指定しなければ、文字定数が正しくコンパイルされません。 Server Express ではコード変換を行ないませんので、コンパイル時に環境変数：LANG の指定が無くても、文字定数はそのままの値でコンパイルされていました。	環境変数：LANG で COBOL ソースのロケールを指定します。 例： export LANG=ja_JP.SJIS
11	コンパイラ	Visual COBOL のバージョン 3.0 より FASTINIT 指令が追加され、デフォルトで 有効になります。FASTINIT 指令が有効の状態で、集団表項目に対する INITIALIZE 処理を行うと、この指令のマニュアルに記載の通り、「第 1 表要素の空白詰め項目、データ ポインター、オブジェクト参照、またはプログラム ポインターの既存の内容が、表の残りの部分に伝播されます」が適用されます。	過去バージョンの製品から移行時に、集団表項目への INITIALIZE 処理結果が異なる場合があります。 この指令は性能を改善するために追加されたものですが、このような事象が発生する場合があります。Visual COBOL 3.0 より前の製品・バージョンと同様の結果を期待する場合は、NOFASTINIT 指令を指定します。
12	コンパイラ	9.0 より、新たに DECLARE 指令が導入されました。デフォルトでは、SECTION 内で DECLARE 句を用いて宣言された項目を呼び出し毎に固有の領域を割り当てます。一方、以前のバージョンでは、呼び出しに対して常に同じ領域を割り当てていました。	SECTION 呼び出し時における DECLARE 句を利用した項目値の管理方法が異なるため、ロジックによっては、9.0 以前と異なる結果になります。 以前のバージョンと同じ結果にするには、コンパイラ指令 DECLARE"1" を指定してください。
13	コンパイラ	Visual COBOL はコンパイル時にプログラムソースの文字コードを UTF-8 に変換していますが、1 行を 256 バイトとして扱っています。UTF-8 は日本語を 1 文字が 3 バイト以上で表すため、実際の文字数とバイト長は一致しません。変換後のバイト長が 256 を超える場合は COBCH1744 のエラーが報告されます。	コメント内でエラーが報告されている場合は、実際の動作に影響しないため、対策不要です。 一方、ソース行内でエラーが報告されている場合は、複数行に分割してエラーを解消する必要があります。

6.3. 付属ユーティリティに加えられた改善・改修により顕在化された主な事象

No.	コンポーネント	変更概要	変更による主な影響／対策例
1	MFSORT	256 バイトを超える行を含むコマンドファイルを使ってソートユーティリティを実行するとエラーが返される。 MFSORT ユーティリティはコマンドファイル中に記載された各行の 256 バイト分のコマンドのみを正確に解釈するよう設計されています。旧製品 5.1 以前ではこの制限に該当するコマンドファイルも受け付け設計の範囲を超えた指定が可能でした。旧製品 5.1、および、Visual COBOL ではこの制限を検出する機能が追加され、設計の範囲内で動作するよう徹底されています。本制限についてはマニュアルでも公開しています。 なお、Visual COBOL 3.0 以降では 256 以上を継続行として取り扱うように改善されました。	Visual COBOL 3.0 以前の製品に移行される場合は、コマンドファイル中にある 256 バイトを超える行を改行で区切り、全ての行が 256 バイトに収まるよう調整します。

6.4. ミドルウェアバージョンアップ時におけるミドルウェア側の留意事項

No.	ミドルウェア	該当条件	留意事項
1	Oracle Database Oracle Database Client	下記の条件を全て満たす場合は留意する必要があります。 > エンディアンの変更を伴う移行 （例：Sparc Solaris から Intel Linux への移行） > 8i より前 → 8i 以降へのバージョンアップ	Oracle はホスト変数に CPU ネイティブなバイトオーダーを必要としているようです。 Oracle のサンプルも含め多くの場合 COMP 型の変数をホスト変数として利用しますが、この COMP 型の変数は CPU に関わらずビッグエンディアンによるバイトオーダーで値を保持します。そのため、COMP 型のホスト変数を利用するプログラムをリトルエンディアン環境に移行する際はリトルエンディアンで値を保持させるよう変更が必要となります。この救済策として Oracle は「COMP5」というプリコンパイラ指令（デフォルトは有効）を用意し、プリコンパイル時に COMP 型の変数を CPU ネイティブなバイトオーダーで値を保持する COMP-5 型へ変更させるようです。しかし、この有効範囲はプログラム全体となるため、EXEC SQL と END-EXEC で囲んだホスト変数のみならずそれ以外の変数も COMP-5 に変更してしまいます。そこで、Oracle はこの変換の範囲を EXEC SQL と END-EXEC に囲まれた範囲に限定させる Pro*COBOL オプション「DECLARE_SECTION」も用意しています。しかし、本オプションは「MODE」指令がデフォルトの「ORACLE」の場合、無効となります。 従いまして、「MODE」指令をデフォルト値から変更せず且つ COMP 型から COMP-5 型への変更をホスト変数に限定させる場合は、明示的に「DECLARE_SECTION」を有効にします。
2	Oracle Database Oracle Database Client	下記の条件を全て満たす場合は留意する必要があります。 > Oracle 9i 以降への移行 > Oracle Pro*COBOL で開発 > Windows OS をターゲットに開発 > 動的ロード形式の COBOL アプリケーションを開発	Oracle は Oracle 9i 以降、orasql8.dll に加えて orasql< バージョン番号 >.dll を提供しています。Pro*COBOL が利用する API は orasql8.dll ではなく orasql< バージョン番号 >.dll の方に含まれるようです。しかし、Oracle Pro*COBOL は埋め込み SQL 文をプリコンパイルする際、最初の SQL 文展開の先頭に CALL "ORASQL8" を挿入しているため、そのまま実行すると orasql8.dll がロードされ正しく API を参照できません。orasql< バージョン番号 >.dll を正しくロードできるよう、orasql8.dll をバックアップした上で orasql< バージョン番号 >.dll を orasql8.dll へコピーすることで COBOL アプリケーションより正しく Oracle の API を利用できるようになります。
3	Oracle Database Oracle Database Client Db2	下記の条件を全て満たす場合は留意する必要があります。 > 移行前は Oracle Pro*COBOL、IBM Db2 が提供するプリコンパイラ、DB2 ECM を利用 > 移行後は OpenESQL を利用し、ODBC、ADO.NET、JDBC 経由でデータベースへアクセス	OpenESQL は COBOL 製品が提供する特定のデータベースへ依存しない統合化プリプロセッサです。一部 Oracle や Db2 の SQL 方言も解釈するよう機能拡張されていますが、汎用的なデータベースアクセスを目的にしていることから、基本的には ANSI 標準の SQL 文がサポートされます。OpenESQL は CHECK という指令を用意しており、コンパイル時に実際にデータベースに SQL 文を渡し妥当性を検証できます。コンパイル時に本指令も併せて指定することで、ある程度 OpenESQL で利用できない SQL 文を検出することが可能です。 また、ADO.NET 経由でアクセスする場合に限り、Oracle Pro*COBOL との互換機能を追加で提供する PROCOB という指令が利用可能です。
4	Oracle Database Oracle Database Client	下記の条件を全て満たす場合は留意する必要があります。 > Oracle 12c へバージョンアップ > OpenESQL - ADO.NET で Oracle へ接続	Oracle Data Provider for .NET は 12c より Managed Driver と Unmanaged Driver という 2 つの Driver を用意するようになりました。このうち Unmanaged Driver は 11g 以前の Oracle Data Provider for .NET が提供していた ADO.NET の Driver の後継版となります。こちらについては OpenESQL は従来よりサポートしてきましたが、12c より新たに搭載された Managed Driver については Visual COBOL 2.3 Update 1 よりサポートされています。
5	Db2	Db2 が提供するプリコンパイラを使用して COBOL アプリケーションを開発する場合は留意する必要があります。	Db2 ではインストーラーよっては libdb2gmf.a 等の COBOL が Db2 連携する際に必要なライブラリが typical インストールに含まれないようです。この場合、これらのライブラリは IBM Software Development Kit に含まれるようです。（日本語版では「基本アプリケーション開発ツール」と表示されることもあるようです。）そのため、これらのコンポーネントがデフォルトのインストールとは別に用意されている場合は、インストールの際正しくインストールされるよう明示的に指定します。

No.	ミドルウェア	該当条件	留意事項
6	JBoss Application Server / Enterprise Application Platform	下記の条件を全て満たす場合は留意する必要があります。 > JBoss Application Server / Enterprise Application Platform をバージョンアップ > JBoss 上の Java EE アプリケーションより JCA の仕組みを利用し Enterprise Server 上にディプロイした COBOL アプリケーションへアクセス	製品が提供するリソースアダプターを JBoss へディプロイする場合、JBoss のバージョンによっては JBoss の構成ファイルを調整する必要があります。調整内容については製品マニュアル上でご案内していますため、ご利用の際はそちらをご参照ください。

7. 他の製品・バージョンへの移行をご検討中のお客様

7.1. Server Express, Net Express 製品をご利用中のお客様

Server Express, Net Express 5.x 製品への移行
Visual COBOL 3.0 製品への移行
Visual COBOL 4.0 製品への移行
Visual COBOL 5.0 製品への移行
Visual COBOL 6.0 製品への移行
Visual COBOL 7.0 製品への移行
Visual COBOL 8.0 製品への移行
Visual COBOL 9.0 製品への移行
Visual COBOL 10.0 製品への移行
Visual COBOL 11.0 製品への移行

7.2. Visual COBOL 製品をご利用中のお客様

Visual COBOL 3.0 製品への移行
Visual COBOL 4.0 製品への移行
Visual COBOL 5.0 製品への移行
Visual COBOL 6.0 製品への移行
Visual COBOL 7.0 製品への移行
Visual COBOL 8.0 製品への移行
Visual COBOL 9.0 製品への移行
Visual COBOL 10.0 製品への移行

弊社ウェブサイトのこちらから最新版をダウンロードできます。