専用のシステム/ツールおよびユーティリティ

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

専用のシステム/ツールおよびユーティリティ
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

専用のツールおよびユーティリティの概要

ブラウザ・ベースの管理ポータルは、InterSystems IRIS® インスタンスを管理するための標準ツールですが、他のツールも利用できます。このページは、InterSystems IRIS システムの管理、カスタマイズ、および機能拡張に使用できるいくつかの専用のツールとユーティリティのガイドですが、専門的すぎて、一般的な読者には馴染みの薄い内容になっています。以下に示す理由の 1 つ以上により、他のドキュメントには記載されていません。

- ごく少数のユーザのみが使用している。
- 必須ではなく、使用されるのは通常一度だけ。
- 高度なカスタマイズや拡張機能が可能で、InterSystems IRIS システムについて専門的な知識が必要である。
- レガシー・アプリケーションであるが、現在のシステム上で有用な場合がある。

このページの残りの部分では、このドキュメントで説明されているツールの概要を示します。残りの記事では、これらのツールについて詳細に説明します。これには以下が含まれます。

- "InterSystems IRIS システムのカスタマイズ" では、標準の InterSystems IRIS インストールをカスタマイズして拡張するさまざまな方法を説明します。
- "InterSystems IRIS のリモート管理" では、文字ベースのターミナルや外部プログラムから InterSystems IRIS を管理するためのルーチンとユーティリティについて説明します。
- "プロセス管理" では、個々のプロセスと他の並行プロセスとの間でシステム・リソースの競合が発生する仕組みを管理するためのメソッドとユーティリティについて説明します。
- "cvendian を使用したバイト・オーダー変換" では、ビッグ・エンディアン・プラットフォームとリトル・エンディアン・プラットフォームの間の移行に使用できるツールについて説明します。

これらのツールを使用する場合、以下のドキュメントが役に立つ場合があります。

- "システム管理ガイド" では、InterSystems IRIS システムを管理するための標準のツールと手順について説明します。
- "ターミナルの使用法" では、ObjectScript コマンド行インタフェースの使用方法について説明します。
- "インストール・ガイド" では、ご使用のシステムに InterSystems IRIS をインストールする標準手順について説明します。
- "InterSystems IRIS 監視ガイド" では、InterSystems IRIS を監視するために利用できるツール、ルーチン、およびサードパーティのインタフェースについて説明します。
- "インターシステムズのセキュリティについて" では、認証、承認、監査、データベース暗号化、SSL/TLS など、インターシステムズのセキュリティにおけるさまざまな機能について説明します。

1.1 カスタマイズ

“InterSystems IRIS システムのカスタマイズ”に関する記事では、標準の InterSystems IRIS インストールをカスタマイズして拡張するさまざまな方法を説明します。以下のツールについて説明します。

- 各国言語サポートのシステム・クラス – 各国言語サポート (NLS) と呼ばれるローカライズのサポートについて説明します。これによってアプリケーションは、エンジニアリングの変更なしにさまざまな言語や地域に適応できます。
- InterSystems IRIS ^%ZSTART および ^%ZSTOP ルーチンの使用法 – カスタマイズされたシステム・ルーチンを作成する方法について説明します。そのルーチンは、システム・イベント検出時に自動的に呼び出されます。ルーチンを呼び出せるのは、InterSystems IRIS が起動または停止したとき、ユーザがログインまたはログアウトを実行したとき、JOB が開始または終了したとき、あるいは外部プログラムが CALLIN を開始または完了したときです。
- %ZLang による ObjectScript の拡張 – %ZLANG 言語拡張ライブラリを使用して、ObjectScript にカスタム機能を追加する方法について説明します。InterSystems IRIS のインストールに適したユーザ定義のコマンド、関数、およびシステム共通の特殊変数を宣言することができます。これらのカスタム拡張機能は、ObjectScript の標準機能とまったく同じように動作します。

1.2 システムのリモート管理

“InterSystems IRIS のリモート管理”に関する記事では、ターミナルや外部プログラムから InterSystems IRIS を管理するためのルーチンとユーティリティについて説明します。以下のツールについて説明します。

- グローバルの高速コピーのための ^GBLOCKCOPY の使用法 – データベース間でグローバルのコピーを高速に行うルーチンについて説明します。このルーチンは、移行中にデータベースの変換に使用できます。これはターミナルと対話形式で実行するか、バックグラウンドのジョブとして 1 つ以上のグローバルをコピーするバッチに構成されます。組み込みのモニタと、複数のグローバル・コピーの進行を記録するための複数のリポートが含まれます。システム障害が発生した場合、中断した箇所で再開できます。
- スイッチの使用法 – インスタンス単位のフラグで、さまざまな目的に使用できる InterSystems IRIS スイッチについて説明します。特に、バックアップの実行中やクラッシュしたシステムの回復中に、各種システム・プロセスを抑制する際に便利です。
- Windows クライアントからの InterSystems IRIS の制御 – 外部プログラムが Windows クライアントとして機能できるようにする DLL について説明します。Windows クライアントでは、InterSystems IRIS 構成を制御し、InterSystems IRIS プロセスを適切な設定で開始し、指定された構成名の InterSystems IRIS サービス名とディレクトリ・パスを検索し、InterSystems IRIS システムの状態を取得できます。
- 文字ベースの管理ルーチン – COM ポートのリスト、追加、編集、および削除を可能にする ^LEGACYNETWORK ルーチンについて説明します。

1.3 バイト・オーダー変換

記事 ″cvendian を使用したバイト・オーダー変換″ では、ビッグ・エンディアン・プラットフォームとリトル・エンディアン・プラットフォームの間で移行するために、InterSystems IRIS データベースのバイト・オーダーを変換するユーティリティについて説明します。また、指定されたデータベースのバイト・オーダーに関して報告するためのオプションも提供しています。

2

各国言語サポートのシステム・クラスの使用法

最新のアプリケーションは、余分なエンジニアリングなしにさまざまな言語や地域に適応できます。このプロセスはインターナショナライゼーションと呼ばれます。この目的のために、特定のコンポーネントを追加して、特定の地域や言語に対してアプリケーションを適応させるプロセスをローカライズといいます。

ユーザの言語、国、およびその他の特殊な可変情報を定義する一連のパラメータは、ロケールです。ロケールは、入力、出力、およびデータ処理に関する規約を指定します。これには以下が含まれます。

- 数の形式
- 日付と時刻の形式
- 通貨記号
- 文字のソート順
- その他の文字セットへの文字列の自動変換

ロケールを指定する場合は、一般に、使用言語と地域（あるいは他のバリエーション）を記述します。これらには、通常、ISO (International Standards Organization) の言語と地域の略称を指定します。例えば、en-us は米国で使用される英語を示し、en-gb は英国で使用される英語を示します。

注釈　1 つのミラーのすべてのメンバ上の InterSystems IRIS インスタンスは、同一のロケールと照合を持っている必要があります。詳細は、″高可用性ガイド″ の “ミラーリング” の章を参照してください。

2.1 %SYS.NLS のクラス

InterSystems IRIS では、**%SYS.NLS** (NLS は各国言語サポートのことです) パッケージのクラスによるローカライズをサポートしています。これらのクラスには、インターナショナライズされたプログラムをその実行環境に適応させるために InterSystems IRIS が必要な情報が含まれます。このセクションでは、オプションを簡単に説明します。詳細は、各クラスのクラス・ドキュメントを参照してください。

注釈　これらのクラスを使用すると、システムまたはプロセスに現在設定されている値を取得できます。プロセスに関連付けられている値を変更すると、その変更は直ちに反映されます。システム設定を変更するには、アプリケーションで適切な値を指定して新しいロケールを定義し、そのロケールを使用して起動するよう InterSystems IRIS に指示する必要があります。

2.1.1 %SYS.NLS.Locale

%SYS.NLS.Locale のプロパティには、参照が必要になる場合がある現在のロケールに関する情報があります。これらの値を変更しても、システムの動作には影響しません。

2.1.2 %SYS.NLS.Device

%SYS.NLS.Device クラスには、現在のデバイスのプロパティの一部が格納されます。ただし、対象となるデバイスが、オブジェクトをインスタンス化した際に使用していたデバイスであるとは限りません。

一般に、デバイスのプロパティは、そのデバイスが開いたときに設定されます。これにより正しい変換の使用が保証されます。変換テーブルは、デバイスが開いた後に変更することもできます。それには、このクラスのプロセス・インスタンスの XLTTable プロパティを変更しますが、明確な理由がない限り、このプロパティを変更することはお勧めできません。

%SYS.NLS.Device にある他のプロパティにより、変換中に発生したエラーを処理できます。既定では、現在のテーブルで文字を処理できない場合、エラーはトリガされず、問題の発生した文字が疑問符 (?) に変換されます。置換値または置換文字列と呼ばれるこの文字は、他の任意の文字列に変換できます。さらに、未定義の文字を暗黙的に変換する代わりに、エラーを発行させることもできます。この動作は 既定のアクション と呼ばれ、以下から選択できます。

- 0 ー エラー生成
- 1 ー 変換不能な文字を置換値に置換
- 2 ー エラーを無視して変換不能な文字を渡す

このクラスのプロパティの入力および出力処理には、それぞれ次のような個別のプロパティがあります。

- InpDefaultAction
- InpReplacementValue
- OutDefaultAction
- OutReplacementValue

2.1.3 %SYS.NLS.Format

%SYS.NLS.Format クラスには、$ZDATE() や関連する関数の動作に影響を与えるプロパティがあります。これらのプロパティの値は、現在のロケールに定義されている値から継承されますが、他のユーザに影響を与えずにプロセス・レベルで変更することも可能です。例えば、DateSeparator および TimeSeparator プロパティにはそれぞれ、日付項目と時間項目を区切る文字が保持されます。

これらの値を変更した場合の影響については、$ZDATE、$ZDATEH、および $FNUMBER のドキュメントの説明を参照してください。

Locale プロパティ

%SYS.NLS.Format クラスの Locale プロパティでは、現在のプロセスでの値の“外観”を制御できます。以下はその例です。

- Locale が空の文字列の場合は、システムの既定の形式 (通常はアメリカ英語) が適用されます。
- Locale がロケール名 (`rusw` や `csy8` など) の場合は、指定したロケールの形式を使用します。
- Locale が `Current` の場合は、システムの形式を使用します。

このプロパティは、オブジェクトをインスタンス化した後、または以下のように目的のロケールを %New() メソッドに渡すことにより変更できます。

ObjectScript

```
Set fmt = ##class(%SYS.NLS.Format).%New("jpnw")
```

これらの変更は、現在のプロセスにのみ影響します。

2.1.4 %SYS.NLS.Table

%SYS.NLS.Table クラスは、システムの既定を示すオブジェクトや、テーブルのさまざまなカテゴリに対するプロセスの現在の設定を示すオブジェクトをインスタンス化します。NLS の基本的なメカニズムであるテーブルでは、アプリケーション・データを入力として受け入れ、順序付けして、指定のロケールに適した形式で表示することができます。%SYS.NLS.Locale では、システム・オブジェクトのプロパティを変更してもシステムに影響を与えません。ただし、プロセス・オブジェクトからプロパティを変更すると、関連付けられている動作が直ちに変更されます。

NLS のテーブルは、入出力テーブルと内部テーブルに分類できます。各テーブル・タイプには、以下に示すような、独自の関連データがあります。

入出力テーブル

これらのテーブルは、システムが稼動する現在のロケールでサポートされている基本文字セットと、InterSystems IRIS 以外のエンティティでサポートされている他言語文字セットを変換します。例えば、ロケールの文字セットが Latin2 (より正確には ISO 8859-2) で、他言語文字セットが、ターミナルとの通信に一般に使用される UTF-8 だとします。この場合、出力には Latin2-to-UTF8 (Latin2 を UTF8 へ変換する) などのテーブルが使用され、入力には UTF8-to-Latin2 (UTF8 を Latin2 へ変換する) などの逆マッピング・テーブルが必要になります。

ここでは (入力用と出力用に) 2 つのテーブルを使用していますが、通常、これらのテーブルは互いを補完します。InterSystems IRIS では、処理を簡潔にするため、ロケール定義とシステムの既定については、1 組みの入出力テーブルに対して 1 つの名前を使用します。この名前には、通常、他言語文字セットの名前が使用されますが、暗黙の前提として、もう 1 つの名前はロケールの文字セットを使用した名前になります。ただし、カスタム・テーブルを作成する場合は、変換が行われることを示す任意の名前を選択することができます。

入出力テーブルはデバイスで使用されます。この場合、デバイスという単語は、InterSystems IRIS が外部とやり取りするインタフェースを示します。プロセス・インタフェースとシステム呼び出しインタフェースを含めて、このインタフェースでは変換が必要になります。

- ターミナル
- その他のターミナル接続
- 外部ファイル
- TCP/IP 接続
- プリンタ
- InterSystems IRIS プロセス
- システム呼び出し

内部テーブル

内部テーブルでも、現在のロケールの文字セットで構成される文字列を他の値にマップしますが、このテーブルは外部との通信を目的とはしていません。内部テーブルは、以下の文字を識別します。

- パターン・マッチング

  文字、数字、句読点など、特定のパターン・コードに一致する文字を識別します。

- 識別子

  識別子テーブルは、識別子に使用できる文字を指定します。

- 大文字のアルファベット、小文字のアルファベット、およびタイトルで使用される大文字。

これらは構造的に入出力テーブルと似ています。このテーブルは、ある 1 つの文字セットを、偶然同じである別の文字セットにマップします。ただし、これらのテーブルは、入出力処理ではなく、$ZCONVERT() のコンテキストで使用されます。

- 照合順序

  これらのテーブルは、ある文字列を、グローバル添え字での使用に適したその文字列の内部表現にマップします。言語が異なれば、ディクショナリ順序での照合ルールは異なります。これらのルールは照合テーブルにカプセル化されています。

- $X/$Y アクション

  これらのテーブルでは、文字を、特殊変数 $X および $Y とどのように関係するのかを示す値にマップします。文字を出力した後に $X および $Y を増分するのか、文字を印刷可能にするかどうかなど、$X/$Y テーブルにはこれらの答えが含まれます。

注釈　InterSystems IRIS のすべてのバージョンで使用可能な照合のリストは固定されています。既存の照合がニーズに適合しない場合は、インターシステムズのサポート窓口までお問い合わせください。

2.2 %SYS.NLS の使用例

重要　これらの例はすべて実行可能ですが、いずれの例にも [**実行する**] ボタンはありません。これらは現在のロケールに対するプロセス既定値を操作するからです。また、多くが管理特権や %SYS ネームスペースへの書き込みアクセス権を必要とします。これらの例を実行する場合は、インターシステムズのターミナル機能 (Windows) や TCP/IP 接続を介して、かつ適切な特権を持ったユーザとして、別のプロセスで実行してください。

2.2.1 現在のロケール情報の表示

この例では、現在のシステム・ロケールに関する情報が表示されます。

ObjectScript

```
Set Info = ##class(%SYS.NLS.Locale).%New()
Set Items = "Name" _
            "/Description" _
            "/Country" _
            "/CountryAbbr" _
            "/Language" _
            "/LanguageAbbr" _
            "/Currency" _
            "/CharacterSet"

Write !
For i = 1 : 1 : $LENGTH(Items, "/")
{
  Set Item = $PIECE(Items, "/", i)
  Write $JUSTIFY(Item, 15),": ", $PROPERTY(Info, Item), !
}
```

2.2.2 システムおよびプロセス・テーブル・データの表示

この例を実行する前に一部のプロパティが外部で変更されている場合を除き、この例では、システム・テーブルとプロセス・テーブルに同じ値が表示されます。

ObjectScript

```
Set IOTables = "Process" _
               "/IRISTerminal" _
               "/OtherTerminal" _
               "/File" _
               "/TCPIP" _
               "/SystemCall" _
               "/Printer"
Set IntTables = "PatternMatch" _
                "/Identifier" _
                "/Uppercase" _
                "/Lowercase" _
                "/Titlecase" _
                "/Collation" _
                "/XYAction"

// iterate over the systems, and then the process data
For Type = "System", "Process"
{
  Write !
  Set Table = ##class(%SYS.NLS.Table).%New(Type)
  Write "Type: ", Type, !

  Write "I/O Tables", !
  For i = 1 : 1 : $LENGTH(IOTables, "/")
  {
    Set PropName = $PIECE(IOTables, "/", i)
    Write $JUSTIFY(PropName, 15), ": ", $PROPERTY(Table, PropName), !
  }

  Write "Internal Tables", !
  For i = 1 : 1 : $LENGTH(IntTables, "/")
  {
    Set PropName = $PIECE(IntTables, "/", i)
    Write $JUSTIFY(PropName, 15), ": ", $PROPERTY(Table, PropName), !
  }
}
```

2.2.3 日付と時刻の表示の変更

%SYS.NLS.Format クラスには、例えば日付項目と時間項目を区切る文字をそれぞれに保持する、DateSeparator および TimeSeparator プロパティが含まれます。米国の既定のロケールである enu8 (Unicode システムでは enuw) では、これらはそれぞれ、スラッシュ文字 (/) とコロン (:) になります。これらの変更方法について以下に例を示します。

ObjectScript

```
// display the current defaults
// date is 10 April 2005
// time is 6 minutes 40 seconds after 11 in the morning
Write $ZDATE("60000,40000"), !

// now change the separators and display it again
Set fmt = ##class(%SYS.NLS.Format).%New()
Set fmt.DateSeparator = "_"
Set fmt.TimeSeparator = "^"
Write !, $ZDATE("60000,40000")
```

以下に示すこの例では、月の名前がアルファベットの連続する文字に変換されます (デモンストレーション用)。このためには、MonthName プロパティに、スペースで区切った月名一覧を設定します。リストの先頭文字はスペースです。

ObjectScript

```
// get the format class instance
Set fmt = ##class(%SYS.NLS.Format).%New()

// define the month names
Set Names = " AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK LLL"
Set fmt.MonthAbbr = Names
Set rtn = ##class(%SYS.NLS.Format).SetFormatItem("DATEFORMAT", 2)

// show the result
Write $ZDATE(60000, 2)
```

2.2.4 数字の表示方法の変更

%SYS.NLS.Format 内の一部のプロパティは、$Number() が数字を解釈する方法を制御します。英語のロケールでは、整数と小数点以下の数字の区切りに小数点が使用され、数字を 3 桁ずつ区切るためにコンマが使用されています。これも変更可能です。

ObjectScript

```
// give the baseline display
Write $Number("123,456.78"), !

Set fmt = ##class(%SYS.NLS.Format).%New()
// use "/" for groups of digits
Set fmt.NumericGroupSeparator = "."

// group digits in blocks of 4
Set fmt.NumericGroupSize = 4

// use ":" for separating integer and fractional parts
Set fmt.DecimalSeparator = ","

// try interpreting again
Write $Number("12.3456,78"), !
```

2.2.5 ファイルの変換の設定

以下の例は、ファイルに書き込まれたデータ表現をアプリケーションで制御しています。

ObjectScript

```
// show the process default translation (RAW, no translation performed)
Set Tbl = ##class(%SYS.NLS.Table).%New("Process")
Write "Process default translation: ", Tbl.File, !

// create and open a temporary file
// use XML for the translation
Set TempName = ##class(%Library.File).TempFilename("log")
Set TempFile = ##class(%Library.File).%New(TempName)
Do TempFile.Open("WSNK\XML\")
Write "Temp file: ", TempFile.CanonicalNameGet(), !

// write a few characters to show the translation
// then close it
Do TempFile.WriteLine(("--" _ $CHAR(38) _ "--"))
Do TempFile.Close()

// now re-open it in raw mode and show content
Do TempFile.Open("RSK\RAW\")
Do TempFile.Rewind()
Set MaxChars = 50
Set Line = TempFile.Read(.MaxChars)
Write "Contents: """, Line, """", !

// finish
Do TempFile.Close()
Do ##class(%Library.File).Delete(TempName)
Set TempFile = ""
```

変換テーブルの詳細は、$ZCONVERT 関数のドキュメントにある“3 つのパラメータ形式：エンコード変換”のセクションを参照してください。

2.3 Config.NLS クラス

%SYS.NLS はあらゆる場所で使用可能であり、一般的な使用を目的としていますが、これとは対照的に、Config.NLS のクラスは、%SYS ネームスペースでのみ、管理者権限を持っているユーザのみが使用できます。通常、カスタムのロケー

ルおよびテーブルを作成する必要がある管理者は、管理ポータルで NLS ページを使用します。特別な要求を持つユーザを除いては、Config.NLS を使用する必要はありません。

Config.NLS パッケージには、以下の 3 つのクラスがあります。

- Locales – 国または地理的地域に対するすべての定義および既定値を格納します。
- Tables – テーブルに関する高レベルの説明を格納しますが、マッピング自体は格納しません。
- SubTables – 固有の文字のマッピングを格納します。複数のテーブルで共有することができます。

Tables クラスと SubTables クラスが分かれている主な理由は、データの重複を避けるためです。異なる文字セットに対する Tables が、同一のマッピングを共有する (したがって、同一の SubTable を共有する) こともあります。また、Tables のクラスでは既定のアクションと置換値を定義します (上記の %SYS.NLS 内のこれらのプロパティに関する説明を参照)。したがって、別個の Tables が、同一の SubTable を共有している場合でも、これらの属性が異なっている場合があります。このような柔軟性があるため、Tables と SubTables の間の関係を適切に管理することは多少複雑になりますが、メリットも大きいので無駄ではありません。すべてのハウスキーピング処理が行われている管理ポータルと %SYS.NLS クラスでは、Tables と SubTables が分かれていることはユーザからはわかりません。ただし、Config.NLS を使用する際には、これを明示的に行う必要があります。

2.3.1 ユーザ定義のロケールおよびテーブルの名前付け規約

カスタム項目とシステム項目とを区別して、アップグレードを簡単にするには、項目名の先頭に y を使用します。例えば、`XLT-yEBCDIC-Latin1` や `XLT-Latin1-yEBCDIC` のようにします。

注意　この規約に準拠しないユーザ定義のテーブル、サブテーブルおよびロケールは、システムのアップグレード時に削除される場合があります。これを避けるには、ユーザ定義のテーブルとロケールを XML ファイルにエクスポートし、アップグレード後に再インポートします。

インターシステムズの SubTable をコピーして、カスタムの SubTable を作成した場合、このタスクを実行するユーティリティは、自動的に同じ名前を使用し、数字の接尾辞を付加します。したがって、Latin2-to-Unicode SubTable のコピーの名前は、`XLT-Latin2-Unicode.0001` および `XLT-Unicode-Latin2.0001` のようになります。

2.4 Config.NLS の使用の例

このセクションでは、以下の例について説明します。

- 使用可能なロケールのリスト
- 特定のロケールのテーブルのリスト
- カスタム・ロケールの作成

2.4.1 使用可能なロケールのリスト

この例では、Config.NLS.Locales の List() クラス・クエリを使用し、使用可能なロケール識別子と説明のリストを表示します。

ObjectScript

```
  new $namespace
  set $namespace="%SYS"

  set stmt=##class(%SQL.Statement).%New()
  set status=stmt.%PrepareClassQuery("Config.NLS.Locales","List")
  if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

  set locales=stmt.%Execute("*")
  if (locales.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", locales.%SQLCODE, ": ",
locales.%Message quit}

  while locales.%Next()
  {
    write locales.%Get("Name"), " - ", locales.%Get("Description"), !
  }
  if (locales.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", locales.%SQLCODE, ": ",
locales.%Message quit}
```

2.4.2 特定のロケールのテーブルのリスト

以下の例では、米国の英語 (使用可能な場合) に対する Unicode ロケールを構成するテーブルを示します。

ObjectScript

```
  new $namespace
  set $namespace="%SYS"

  // establish the locale identifier, try
  // United States - English - Unicode
  // United States - English - 8-bit
  Set Loc = "enuw"
  Do ##class(Config.NLS.Locales).Exists(Loc, .Ref, .Code)
  If (##class(%SYSTEM.Status).IsError(Code))
  {
    Set Loc = "enu8"
    Do ##class(Config.NLS.Locales).Exists(Loc, .Ref, .Code)
    If (##class(%SYSTEM.Status).IsError(Code))
    {
      Do ##class(%SYSTEM.Status).DisplayError(Code)
      Quit
    }
  }

  // get the local array of table names
  Write "Tables for locale: ", Loc, !
  Do Ref.GetTables(.Tables)
  Set Type = $ORDER(Tables(""))
  While (Type '= "")
  {
    Set Name = $ORDER(Tables(Type, ""))
    While (Name '= "")
    {
      Set Mod = $ORDER(Tables(Type, Name, ""))
      While (Mod '= "")
      {
        Write Type, " - ", Name, " - ", Mod, !
        Set Mod = $ORDER(Tables(Type, Name, Mod))
      }
      Set Name = $ORDER(Tables(Type, Name))
    }
    Set Type = $ORDER(Tables(Type))
  }
```

2.4.3 カスタム・ロケールの作成

この例では、カスタム・テーブルでカスタム・ロケールを作成するためのテンプレートを提供します。このカスタム・テーブルでは、EBCDIC (米国で使用される一般的な形式) と Latin-1 (ISO-8859-1) の間の変換を実行します。詳細は、各クラスのドキュメントを参照してください。

その他のすべてのテーブルについては、最初に、文字のマッピングの定義を取得する必要があります。この例では、http://source.icu-project.org (International Components for Unicode) の Web サイトのデータ・ファイルを使用します。

該当するデータ・ファイルは、シャープ記号 (#) で始まるコメント行の後に、以下の形式の一連の変換定義行が続く、テキスト・ファイルです。

```
<Uuuuu>  \xee |0
```

ファイルの抜粋を以下に示します。

```
#
#UNICODE EBCDIC_US
#_______ _________
<U0000>  \x00 |0
<U0001>  \x01 |0
<U0002>  \x02 |0
<U0003>  \x03 |0
<U0004>  \x37 |0
<U0005>  \x2D |0
...
```

これらの行では、Unicode 文字 Uaaaa が EBCDIC 文字の ¥xbb にマッピングされることを示しています (ここで aaaa および bb は 16 進数の表現です)。このテーブルは逆変換が可能であり、EBCDIC 文字の ¥xbb を Unicode 文字の Uaaaa に再マッピングすることができると仮定しています。これにより、1 回のスキャンで、同じデータ・ファイルから両側 (つまり EBCDIC-to-Latin1 および Latin1-to-EBCDIC) を作成することができます。Unicode の範囲は 0 ～ 255 のみなので、これは実際には Latin-1 テーブルになります。

このプロセスでは、最初に SubTable オブジェクトを作成し、次に Table を作成し、最後に Locale を作成します。最初の手順として、このプロセスでは 2 つの SubTables オブジェクトを作成し、Name および Type プロパティを初期化して、定義ファイルから読み込んだデータで FromTo マッピング配列を埋めます。

SubTable の名前の形式は、Type-FromEncoding-ToEncoding です。通常の I/O 変換の Type は "XLT" であるので、SubTable の名前は `XLT-yEBCDIC-Latin1` および `XLT-yLatin1-EBCDIC` になります。

以下のコードで SubTables オブジェクトを作成します。わかりやすくするために、ここでは省略していますが、実際のプログラムでは、何度もコードの整合性チェックが行われます。この例では、このサンプル・コードを何度も実行できるように、既存の以前のバージョンの同じオブジェクト (SubTables、Tables および Locales) を削除しています。厳密には、クラス・メソッド Exists() を使用して以前のオブジェクトの存在をチェックし、既に存在する場合は、別の処理を実行する必要があります。

ObjectScript

```
// Names for the new SubTables (save for later)
Set nam1 = "XLT-Latin1-yEBCDIC"
Set nam2 = "XLT-yEBCDIC-Latin1"

// Delete existing SubTables instances with same ids
Do ##class(Config.NLS.SubTables).Delete(nam1)
Do ##class(Config.NLS.SubTables).Delete(nam2)

// Create two SubTable objects
Set sub1 = ##class(Config.NLS.SubTables).%New()
Set sub2 = ##class(Config.NLS.SubTables).%New()

// Set Name and Description
Set sub1.Name = nam1
Set sub1.Description = "ICU Latin-1->EBCDIC sub-table"
Set sub2.Name = nam2
Set sub2.Description = "ICU EBCDIC ->Latin-1 sub-table"
```

SubTables には、マルチバイトの変換を実行するかどうかを示す小さな整数である、type プロパティが含まれます。この例では、type は、シングルバイト・マッピングを示す 0 に設定されています。データ・ファイルに定義されていないコード・ポイント (文字) がそれ自体にマッピングされるように、このマッピングは初期化されています。

ObjectScript

```
  // Set Type (single-to-single)
  Set sub1.Type = 0
  Set sub2.Type = 0

  // Initialize FromTo arrays
  For i = 0 : 1 : 255
  {
    Do sub1.FromTo.SetAt(i, i)
    Do sub2.FromTo.SetAt(i, i)
  }
```

次に、アプリケーションがファイルを読み込みます。ファイルの定義が、既定のマッピングとして設定された定義をオーバーライドします。$ZHEX() 関数により、コードが 16 進数から 10 進数に変換されます。

ObjectScript

```
  // Assume file is in the mgr directory
  Set file = "glibc-EBCDIC_US-2.1.2.ucm"

  // Set EOF exit trap
  Set $ZTRAP = "EOF"

  // Make that file the default device
  Open file
  Use file
  For
  {
    Read x
    If x?1"<U"4AN1">".E
    {
      Set uni = $ZHEX($E(x,3,6)),ebcdic = $ZHEX($E(x,12,13))
      Do sub1.FromTo.SetAt(ebcdic,uni)
      Do sub2.FromTo.SetAt(uni,ebcdic)
     }
  }

EOF  // No further data
  Set $ZT = ""
  Close file

  // Save SubTable objects
  Do sub1.%Save()
  Do sub2.%Save()
```

これで文字のマッピングは完了です。次に、定義したばかりの SubTables オブジェクトを参照する Table オブジェクトを作成します。Table オブジェクトは実際には SubTables の記述子であり、プロパティはわずかしかありません。以下のコードによって、この 2 つが関連付けられます。

ObjectScript

```
  // Delete existing Tables instances with same ids
  Do ##class(Config.NLS.SubTables).Delete("XLT", "Latin1", "yEBCDIC")
  Do ##class(Config.NLS.SubTables).Delete("XLT", "yEBCDIC", "Latin1")

  // Create two Table objects
  Set tab1 = ##class(Config.NLS.Tables).%New()
  Set tab2 = ##class(Config.NLS.Tables).%New()

  // Set description
  Set tab1.Description = "ICU loaded Latin-1 -> EBCDIC table"
  Set tab2.Description = "ICU generated EBCDIC -> Latin-1 table"

  // Set From/To encodings
  Set tab1.NameFrom = "Latin1"
  Set tab1.NameTo = "yEBCDIC"
  Set tab2.NameFrom = "yEBCDIC"
  Set tab2.NameTo = "Latin1"

  // Set SubTable
  Set tab1.SubTableName = nam1
  Set tab2.SubTableName = nam2

  // Set Type
  Set tab1.Type = "XLT"
  Set tab2.Type = "XLT"
```

```
// Set Default Action
// 1 = Replace with replacement value
Set tab1.XLTDefaultAction = 1
Set tab2.XLTDefaultAction = 1

// Set Replacement value of "?"
Set tab1.XLTReplacementValue = $ASCII("?")
Set tab2.XLTReplacementValue = $ASCII("?")

// Set Reversibility
// 1 = Reversible
// 2 = Generated
Set tab1.XLTReversibility = 1
Set tab2.XLTReversibility = 2

// Set Translation Type
// 0 = non-modal to non-modal
Set tab1.XLTType = 0
Set tab2.XLTType = 0

// Save Table objects
Do tab1.%Save()
Do tab2.%Save()
```

Tables が定義されたら、構築の最後の手順として、新しいテーブルを組み込むロケール・オブジェクトの定義を行います。アプリケーションで、空の Locale オブジェクトを作成し、各プロパティを埋め込みます。これは、Tables および SubTables の場合と同様です。ただし、Locale の場合はより大きく複雑です。このような単純な変更を行う最も簡単な方法は、既存のロケールをコピーして、必要箇所のみを変更することです。このプロセスでは、ソース・ロケールとして enu8 を使用し、yen8 という新しい名前を付けます。イニシャル y によって、これがカスタム・ロケールであることが明確になり、アップグレード時に削除されなくなります。

ObjectScript

```
// Delete existing Locales instance with the same id
Do ##class(Config.NLS.Locales).Delete("yen8")

// Open source locale
Set oldloc = ##class(Config.NLS.Locales).%OpenId("enu8")

// Create clone
Set newloc = oldloc.%ConstructClone()

// Set new Name and Description
Set newloc.Name = "yen8"
Set newloc.Description = "New locale with EBCDIC table"
```

ロケールが適切に設定されると、プロセスでは、起動時にロードされる I/O テーブルのリストに EBCDIC テーブルを追加します。これは、以下のように、配列プロパティ XLTTables にノードを挿入することによって行われます。

```
XLTTables(<TableName>) = <components>
```

- tableName により、このロケールの入力テーブルと出力テーブルのペアが特定されます。

  この名前は y で始まる必要はないので、EBCDIC を使用します。

- components は、以下のような 4 項目のリストです。

  1. 入力“From”のエンコーディング
  2. 入力“To”のエンコーディング
  3. 出力“From”のエンコーディング
  4. 出力“To”のエンコーディング

以下のコードで、使用可能なロケールのリストにテーブルを追加します。

ObjectScript

```
// Add new table to locale
Set component = $LISTBUILD("yEBCDIC", "Latin1", "Latin1", "yEBCDIC")
Do newloc.XLTTables.SetAt(component, "EBCDIC")
```

InterSystems IRIS でロケールを使用できるようにするには、その内部形式にコンパイルする必要があります。これは、ロケールの検証と呼ばれることもあります。IsValid() クラス・メソッドは、詳細な分析を行い、ロケールが適切に定義されていない場合、人間が読むことのできるメッセージと共に、2 つの配列を返します。1 つはエラーの配列、もう 1 つは警告の配列です。

ObjectScript

```
// Check locale consistency
If '##class(Config.NLS.Locales).IsValid("yen8", .Errors, .Warns)
{
  Write !,"Errors: "
  ZWrite Errors
  Write !,"Warnings: "
  ZWrite Warns
  Quit
}

// Compile new locale
Set status = ##class(Config.NLS.Locales).Compile("yen8")
If (##class(%SYSTEM.Status).IsError(status))
{
  Do $System.OBJ.DisplayError(status)
}
Else
{
  Write !,"Locale yen8 successfully created."
}
```

2.5 %Library.GlobalEdit の使用によるグローバルの照合の設定

新規に作成された InterSystems IRIS グローバルの照合は、グローバルが作成されたデータベースの既定の照合に自動的に設定されます。InterSystems IRIS のインストールにより作成されたデータベースはすべて InterSystems IRIS 標準の照合に設定されます。ただし、USER は InterSystems IRIS と共にインストールされるロケールの既定の照合に設定されます

データベースの作成後、プロパティを編集してデータベースの既定の照合を変更することができます。InterSystems IRIS 標準、ロケールの既定の照合、またはインスタンスにロードされたその他の照合が選択可能です。データベースの既定の照合が一度設定されると、このデータベースで作成されたすべてのグローバルは、この既定の照合で作成されます。

また、InterSystems IRIS では、この動作をオーバーライドしてグローバルのカスタム照合を指定することもできます。これを行うには、%Library.GlobalEdit クラスの Create() メソッドを使用して、希望の照合を指定します。

ObjectScript

```
Set sc = ##class(%Library.GlobalEdit).Create(ns,
                                             global,
                                             collation,
                                             growthblk,
                                             ptrblock,
                                             keep,
                                             journal,
                                             .exists)
```

以下はその説明です。

- ns ー ネームスペースを指定します。"" は現在のネームスペースを示し、^^directoryname は特定のディレクトリを参照します。
- global ー ^cz2 などの、先頭に ^ を含むグローバル名を指定します。

- collation ー 照合を指定します。照合は、サポート対象の照合のうちのどれか 1 つです。
- growthblk ー データの開始ブロックを指定します。
- ptrblk ー ポインタの開始ブロックを指定します。
- keep ー グローバルが削除されるときに、グローバルのディレクトリ・エントリを保持するかどうかを指定します。1 に設定すると、グローバルが削除されるときに照合、保護、およびジャーナル属性を保持します。
- journal ー この引数は関係がなくなったので、無視されます。
- exists ー グローバルが既に存在するかどうかを示す変数を、参照によって指定します。

グローバルが他のグローバルとは異なる照合を必要とする環境では、データベースを異なる照合ごとに設定し、グローバル・マッピングをネームスペースに追加して、各グローバルを必要な照合を持つデータベースにマップすることをお勧めします。このメソッドによって、特別に Create() メソッド呼び出しを使用したアプリケーション・コードを変更することなく、照合を混合して使用することができます。

2.5.1 サポート対象の照合

以下は、CreateGlobal^%DM サブルーチンの collation 引数で使用される、InterSystems IRIS でサポートされている照合です。

- 5 ー InterSystems IRIS 標準
- 10 ー German1
- 11 ー Portuguese1
- 12 ー Polish1
- 13 ー German2
- 14 ー Spanish1
- 15 ー Danish1
- 16 ー Cyrillic1
- 17 ー Greek1
- 18 ー Czech1
- 19 ー Czech2
- 20 ー Portuguese2
- 21 ー Finnish1
- 23 ー Cyrillic2
- 24 ー Polish2
- 27 ー French1
- 28 ー Finnish2
- 29 ー Hungarian1
- 30 ー German3
- 31 ー Polish3
- 32 ー Spanish2
- 33 ー Danish2

- 34 ー Greek2
- 35 ー Finnish3
- 36 ー Lithuanian1
- 41 ー Danish3
- 44 ー Czech3
- 45 ー Hungarian2
- 47 ー Spanish3
- 49 ー Spanish4
- 51 ー Spanish5
- 52 ー Finnish4

注釈　インスタンスにロードされた照合など、類似したリストを参照するには、ターミナル・ウィンドウを開き、**%SYS%** ネームスペースに変更して、DO ^COLLATE.コマンドを入力します。

2.5.2 インストール済みロケールの既定の照合

InterSystems IRIS の新規インストールのロケールでは、既定の照合は常に最新バージョンの照合です。つまり、数字の接尾語が最も大きい値のものになります (前のセクションのリスト参照)。例えば、スペイン語ロケールをインストールした場合、既定の照合は Spanish5 となります。照合の古いバージョンは既存のデータベースとの互換性のためにサポートされています。

InterSystems IRIS インスタンスがアップグレードされた場合、既定の照合は、更新済みロケールが新しい既定を使用しても保持されます。例えば、既存のインスタンスのロケールが既定の照合として Finnish3 を使用しており、更新済みインスタンスが Finnish4 を使用する場合、アップグレードは Finnish3 を既定として保存しますが、新規グローバルおよびデータベース用に Finnish4 を使用可能にします。

3

^%ZSTART ルーチンと ^%ZSTOP ルーチンによる開始動作と停止動作のカスタマイズ

InterSystems IRIS では、特定のイベントが発生した際に、カスタム・コードを実行できます。必要とされる手順は以下の 2 つです。

1. ^%ZSTART ルーチン、^%ZSTOP ルーチン、またはそれらの両方を定義します。

   注釈　^%ZSTART と ^%ZSTOP は、InterSystems IRIS には含まれておらず、ユーザが作成する必要があります。

   それらのルーチンでは、特定の処理が開始または停止した際に実行するサブルーチンを定義できます。

   ^%ZSTART と ^%ZSTOP は、**%SYS** ネームスペースで作成および定義する必要がありますが、既定以外のデータベースにマッピングできます。

2. 管理ポータルを使用して、目的のサブルーチンを呼び出すように InterSystems IRIS を構成します。

具体的には、^%ZSTART ルーチンと ^%ZSTOP ルーチンを定義し、特定の名前を持つサブルーチンを含める場合、処理の開始時または終了時に、システムによって自動的にこれらのサブルーチンが呼び出されます。サブルーチン名は以下のとおりです。

- SYSTEM ― InterSystems IRIS がシステムとして開始または停止する際に実行されます。
- LOGIN ― ユーザが **%Service_Console** サービスまたは **Service_Telnet** サービスを使用してログインまたはログアウトを実行する際に実行されます。
- JOB ― JOB が開始または終了する際に実行されます。
- CALLIN ― 外部プログラムが CALLIN を開始または完了する際に実行されます。

例えば、LOGIN^%ZSTART が定義されていて、管理ポータルを使用してこのサブルーチンを有効にしている場合、ユーザがログインすると、LOGIN^%ZSTART がシステムによって自動的に呼び出されます。

これらのサブルーチンは、複雑な計算をしたり長時間実行するためのものではありません。ネットワーク・アクセスなどのように長期的に計算したり、潜在的に長時間わたって実行すると、呼び出されたルーチンが返されるまで動作の完了が遅れます。この場合、起動までに時間がかかりすぎるために、ユーザのログインに長い時間がかかるか、JOB の処理能力が低下する場合があります。

注釈　これらのサブルーチンは、通常の InterSystems IRIS 操作の一部として呼び出されます。つまり、電源障害など、InterSystems IRIS を異常終了させる外部イベントでは、^%ZSTOP への呼び出しが生成されません。

注釈　システムに ^%ZSTOP が実装され、アプリケーションに 1 つ以上の $HALT ルーチンが実装されている場合、最後の $HALT が HALT コマンドで停止するまで、^%ZSTOP コードは実行されません。$HALT ルーチンが自身の HALT コマンドの発行に失敗すると、^%ZSTOP コードは実行されない可能性があります。

3.1 設計に関する考慮事項

^%ZSTART および ^%ZSTOP の実行環境には、ある程度制約があります。設計者は以下のような点に注意してください。

- ルーチンは ObjectScript で記述しなければなりません。
- ^%ZSTART は基本的に、引数なしの新しいコマンドで開始されるように実行されるため、ユーザのローカル変数を初期化するなどのタスクを実行するためには使用できません。
- 任意のルーチン・エントリ・ポイントが呼び出されたときに、引数として渡される値はありません。さまざまな状況で異なるアルゴリズムが適用できる場合は、呼び出されたエントリ・ポイントはルーチンの外部、つまりグローバルやシステム変数などのデータを調査して、何を実行するかを決定する必要があります。
- あらゆる条件下で、ルーチンがうまく動作することを確認してください。ルーチンは防御的に記述します。つまり、タスクの完了に必要なすべてのリソースがすぐ使用できる場所にあり、可能であれば演算処理が始まる前に予約されることがこれらのルーチンで確認されるようにします。発生したエラーはそのシステム関数の失敗として報告されるので、エラー抑制と処理の観点から設計を考えることが重要です。リソースが見つからない、あるいはエラーが起こる際に適切に回復できない場合、さまざまな結果がもたらされます。これには、InterSystems IRIS の開始の失敗、スタジオなどの主要な機能の動作の不具合やすぐには検知されないようなわずかな影響などが含まれます。これらのルーチンは、細心の注意を払って記述し、シミュレート条件下でデバッグし、プロダクション・システムで使用する前にシミュレート環境下でテストすることを推奨します。
- 以前の呼び出しの間に見つかった条件や、異なるエントリ・ポイントが有効であると想定しないでください。例えば、JOB^%ZSTART への連続呼び出し間に、次の呼び出しが発生する前に前回使用したファイルが削除される可能性もあります。
- 各エントリ・ポイントはタスクを効率的に実行します。タスクの一部が、潜在的に長時間実行するものであれば、ユーザのアプリケーションの別の部分によって後から処理するように、完了に十分な情報をキューに格納することをお勧めします。
- 統計的に使用する目的で、エントリ・ポイントでデータを永続的に保持させる場合、データの保持にはグローバルや外部ファイルなどを使用する必要があります。
- ルーチンを実行する環境についての条件は、最小限にしてください。例えば、このようなルーチンの開発者は、プログラムが常に特定のジョブ番号で実行されるとは予測できません。設計者は、さまざまなエントリ・ポイントが特定の順序で呼び出されることを想定できません。InterSystems IRIS を実装する複数のプロセスを呼び出す順序が、確定的であることはほとんどありません。
- ルーチンは、システムの開始中の特定の時点で呼び出されるとは限りません。開始中のイベントの配列は、リリースのたびに、または再開のたびに変更する可能性があります。
- いくつかの例外を除き、ルーチンでは見つけた情報をそのままにしておく必要があります。例えば、サブルーチン内で、エントリ時と終了時に $IO の値を保存してリストアせずに再代入すると、ほとんど確実にエラーの原因になります。呼び出し元のルーチンは、このような変更があったことを知る方法がありません。また、呼び出し側で実行環境に対する変更を防ぐのは非常に困難です。したがって、呼び出される側のサブルーチンで、システム処理のコンテキストを妨害しなような対策を講じる必要があります。

  この変更不可規則の一般的な例外として、アプリケーション・プログラムやインストール・プログラムに固有の、プロセス・ローカル値は変更できます。例えば、SYSTEM^%ZSTART エントリ・ポイントで、システム全体の既定値を設定します。同様に、アプリケーションのテストのために日付を設定し、月末処理の妥当性を検証することもできます。

- ^%ZSTOP には、リモート・データベース内のグローバルへの参照を含めることができません。^%ZSTOP の呼び出し時に、これらの参照の一部がアクセスできなくなります。
- シャットダウン・プロセス中に SYSTEM^%ZSTOP が呼び出されると、ユーザは JOB コマンドを使用して新しいプロセスを開始できなくなります。既存のプロセスを終了することはできます。
- これらのルーチンが **IRISSYS** と異なるデータベースにマップされている場合、InterSystems IRIS は、**IRISSYS** ではなく、そのデータベースから実行しようとします。当然ながら、InterSystems IRIS は、呼び出しルーチンにそのデータベースへの適切なアクセス権限があることを事前に確認します。そのネームスペースで必要なあらゆるアプリケーション・グローバルとマッピングに対して、ルーチンに適切なアクセス権限があることを保証するのは、管理者の責任となります。
- SYSTEM^%ZSTART と SYSTEM^%ZSTOP は、$USERNAME が **%SYSTEM** に設定され、$ROLES が **%All** に設定された状態で実行されます。コードを別のユーザ名で実行するには、$SYSTEM.Security.Login() を使用して目的の名前を設定してから、カスタム・コードで続行します。SYSTEM^%ZSTART コードで JOB を使用して追加プロセスを起動すると、それらのプロセスは、開始プロセスと同じユーザ名 (およびロール) を継承します。

  注意　^%ZSTART と ^%ZSTOP にあるすべてのエントリ・ポイントは、システム処理の重要ポイントで呼び出され、システム処理、さらにはデータ処理に対しても、広範な影響が及ぶ場合があります。これらのルーチンの指定目的により、この高いレベルの特権が必要になります。このため、これらのエントリ・ポイントで呼び出すことができるコードのすべてが、くまなくテストされていることを確認する必要があります。さらに、ユーザ指定コードが XECUTE を介して、つまり間接的に実行されることを許可しないでください。

- 終了 (つまり停止) するプロセスでは、分散キャッシュ・クラスタ・データ・サーバからの回答を必要とする任意の参照で <FUNCTION> エラーが発生する場合があります。

注釈　アップグレードでは、InterSystems IRIS は、**IRISSYS** データベースにマップされた %Z* ルーチンのみを保持します。また、.INT コードまたは .MAC コードが利用できる場合、これらをリコンパイルします。他のデータベース内でのルーチンの保存は、サイト管理者が判断します。

3.2 %ZSTART および %ZSTOP を有効にする

ルーチンの設計、開発、コンパイルが完了し、テストできるようになっていれば、管理ポータルから個々のエントリ・ポイントを有効にすることができます。**[システム管理]→[構成]→[追加設定]→[開始設定]** の順に選択して [開始設定] ページに移動し、該当する個々の設定を編集します。

- SystemStart、SystemHalt
- ProcessStart、ProcessHalt
- JobStart、JobHalt
- CallinStart、CallinHalt

1 つ以上のエントリ・ポイントを無効にするには、同じプロシージャを使用しますが、値を **[偽]** に変更します。

3.3 ^%ZSTART および ^%ZSTOP のデバッグ

最終的な環境において ^%ZSTART と ^%ZSTOP をデバッグする機会は制限されています。エラーが発生すると、エラーはオペレータのメッセージ・ログに書き込まれます。これは、それらのルーチンが実行している時点での現在のデバイスです。これは管理者用ディレクトリにある messages.log ファイルです。

メッセージは、失敗の原因とエラーが検出された位置を示します。この位置は、プログラム・ロジックまたはフローで実際にエラーが発生した場所とは異なる場合があります。開発者は、提供された情報からエラーの特性と場所を推定し、ルーチンを修正してください。これにより、今後のテストでは、発生するエラーの特性についてより多くの情報を得ることができるようになります。

3.4 ^%ZSTART および ^%ZSTOP の削除

ルーチンを削除する前に、管理ポータル経由でエントリ・ポイントの各種オプションを必ず無効にすることを強くお勧めします。この変更を有効にするために、ポータルが InterSystems IRIS を再起動するように警告した場合は、次に進む前に再起動も実行してください。これにより、削除される間に実行されるエントリ・ポイントがなくなります。

^%ZSTART および ^%ZSTOP (および、サポートする任意のルーチン) が永続的に格納されることを覚えておいてください。これらのトレースをすべて削除するには、管理ポータルから削除してください。

3.5 例

以下は、システムの動作状況を追跡する単純なログの実例です。^%ZSTART と ^%ZSTOP の例を示します。どちらでも、3 番目の例のルーチン (^%ZSSUtil) のサブルーチンが便宜上使用されます。

3.5.1 ^%ZSSUtil の例

このルーチンには、2 つのパブリック・エントリ・ポイントがあります。1 つは、オペレータのメッセージ・ログ・ファイルに単一行を書き込みます。もう 1 つは、ローカル・ログ・ファイルに名前と値の組み合わせのリストを書き込みます。どちらのファイルも管理者用ディレクトリにあり、そのディレクトリは、%Library.File クラスの ManagerDirectory() メソッドにより返されます。

ObjectScript

```
%ZSSUtil ;
    ; this routine packages a set of subroutines
    ; used by the %ZSTART and %ZSTOP entry points
    ;
    ; does not do anything if invoked directly
    quit

#define Empty ""
#define OprLog 1

WriteConsole(LineText) PUBLIC ;
    ; write the line to the messages log
    ; by default the file messages.log in the MGR directory
    new SaveIO

    ; save the current device and open the operator console
    ; set up error handling to cope with errors
    ; there is little to do if an error happens
    set SaveIO = $IO
    set $ZTRAP = "WriteConsoleExit"
    open $$$OprLog
```

```
    use $$$OprLog
    ; we do not need an "!" for line termination
    ; because each WRITE statement becomes its
    ; own console record (implicit end of line)
    write LineText
    ; restore the previous io device
    close $$$OprLog
    ; pick up here in case of an error
WriteConsoleExit ;
    set $ZTRAP = ""
    use SaveIO
    quit

WriteLog(rtnname, entryname, items) PUBLIC ;
    ; write entries into the log file
    ; the log is presumed to be open as
    ; the default output device
    ;
    ; rtnname: distinguishes between ZSTART & ZSTOP
    ; entryname: the name of the entry point we came from
    ; items: a $LIST of name-value pairs
    new ThisIO, ThisLog
    new i, DataString

    ; preserve the existing $IO device reference
    ; set up error handling to cope with errors
    ; there is little to do if an error happens
    set ThisIO = $IO
    set $ZTRAP = "WriteLogExit"

    ; construct the name of the file
    ; use the month and day as part of the name so that
    ; it will create a separate log file each day
    set ThisLog = "ZSS"
                _ "-"
                _ $EXTRACT($ZDATE($HOROLOG, 3), 6, 10)
                _".log"

    ; and change $IO to point to our file
    open ThisLog:"AWS":0
    use ThisLog

    ; now loop over the items writing one line per item pair
    for i = 1 : 2 : $LISTLENGTH(items)
    {
        set DataString = $LISTGET(items, i, "*MISSING*")
        if ($LISTGET(items, (i + 1), $$$Empty) '= $$$Empty)
        {
            set DataString = DataString
                           _ ": "
                           _ $LISTGET(items, (i + 1))
        }
        write $ZDATETIME($HOROLOG, 3, 1),
              ?21, rtnname,
              ?28, entryname,
              ?35, DataString, !
    }

    ; stop using the log file and switch $IO back
    ; to the value saved on entry
    close $IO
    ; pick up here in case of an error
WriteLogExit ;
    set $ZTRAP = ""
    use ThisIO
    quit
```

各ラベルの説明を以下に示します。

^%ZSSUtil

このルーチンは、他のルーチンと同様に、以下のコマンドから呼び出されると良好な結果が得られるように、まず QUIT コマンドを実行します。

ObjectScript

```
    do ^%ZSSUtil
```

#DEFINE 配列は、外観をそろえるために、プログラムの本文に指定された制約を提供します。このインスタンスでは、空文字列とオペレータのメッセージ・ログのデバイス番号を指定します。

WriteConsole^%ZSSUtil

このエントリ・ポイントは非常に単純です。容量の少ない出力用に、またデバッグの出力に使用するための最小限のルーチンとして、設計されたものです。

引数として 1 つの文字列を取り、これをオペレータのメッセージ・ログに出力します。ただし、呼び出し全体にわたり、現在の $IO の接続状態を保持し、リストアするための注意が必要です。

各項目がデバイスに送信された結果、メッセージ・ログには別々のレコードが出力されます。したがって、以下のように、4 つのレコードを出力します。

```
WRITE 1, 2, 3, !
```

最初の 3 つは 1 桁の数字から成り、4 つ目は空白行から成ります。1 行に複数の項目を記述したい場合は、呼び出し元がこれらを文字列に連結させなければなりません。

WriteLog^%ZSSUtil

このサブルーチンは、^%ZSTART または ^%ZSTOP 内の任意のエントリ・ポイントから呼び出すことができます。最初の 2 つの引数により、サブルーチンがどのように起動されたかを報告するために必要な情報を渡します。3 番目の引数は、ログに出力される名前と値の組み合わせの $LIST です。

このエントリ・ポイントは最初に、そこで使用するファイルの名前を作成します。ログ管理を簡素化するため、この名前にはルーチンが入力された月日が含まれます。したがって、このサブルーチンを呼び出すと、現地時間が深夜 12 時を経過するたびにファイルが新規作成されます。そのファイル名は呼び出し時にのみ決定されるため、引数として渡される名前と値の組み合わせはすべて、同じファイルに表示されます。

いったん名前が作成されると、$IO の現在値を後で使用できるように保存し、出力デバイスを指定されたログ・ファイルに切り替えます。OPEN コマンドに使用するパラメータによって、そのファイルがなければ作成するように指定されています。timeout がゼロの場合、InterSystems IRIS がファイルを 1 回だけ開こうとします。もし開くことができなければ失敗します。

そのファイルがいったん開かれると、コードは名前と値の組み合わせをループします。それぞれの組み合わせに対して、呼び出し元ルーチン名およびエントリ・ポイント名が書き込まれ、その行に名前と値の組み合わせが続きます(値の部分が空の文字列の場合は、名前のみが書き込まれます)。組み合わせは 1 行に 1 個ずつ出力されます。読みやすいように、各行の最初の 3 つの値は一列に並ぶようになっています。

すべての組み合わせが出力された後、ログ・ファイルを終了し、先ほど保存した $IO の値がリストアされ、コントロールは呼び出し元に返されます。

3.5.2 ^%ZSTART

このルーチンは、実際に InterSystems IRIS に呼び出されるエントリ・ポイントを含みます。上記の ^%ZSSUtil の機能も使用します。すべてのエントリ・ポイントはほとんど同じように動作し、情報をログに配置します。SYSTEM エントリ・ポイントは、他に比べるとやや複雑で、オペレータ・メッセージ・ログにも情報を配置します。

ObjectScript

```
%ZSTART ; User startup routine.

#define ME "ZSTART"
#define BgnSet "Start"
#define Empty ""

    ; cannot be invoked directly
    quit

SYSTEM ;
    ; InterSystems IRIS starting
```

```
    new EntryPoint, Items

     set EntryPoint = "SYSTEM"

     ; record the fact we got started in the messages log
     do WriteConsole^%ZSSUtil((EntryPoint
                               _ "^%"
                               _ $$$ME
                               _ " called @ "
                               _ $ZDATETIME($HOROLOG, 3)))

    ; log the data accumulate results
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3),
                            "Job", $JOB,
                            "Computer", ##class(%SYS.System).GetNodeName(),
                            "Version", $ZVERSION,
                            "StdIO", $PRINCIPAL,
                            "Namespace", $NAMESPACE,
                            "CurDirPath", ##class(%File).ManagerDirectory(),
                            "CurNSPath", ##class(%File).NormalizeDirectory(""),
                            "CurDevName", $System.Process.CurrentDevice(),
                            "JobType", $System.Process.JobType(),
                            "JobStatus", $ZHEX($ZJOB),
                            "StackFrames", $STACK,
                            "AvailStorage", $STORAGE,
                            "UserName", $System.Process.UserName())
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)

    quit
```

ObjectScript

```
LOGIN ;
    ; a user logs into InterSystems IRIS
    new EntryPoint, Items

    set EntryPoint = "LOGIN"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit

JOB ;
    ; JOB'd process begins
    new EntryPoint, Items

     set EntryPoint = "JOB"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit

CALLIN ;
    ; a process enters via CALLIN interface
    new EntryPoint, Items

     set EntryPoint = "CALLIN"
     set Items = $LISTBUILD($$$BgnSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil($$$ME, EntryPoint, Items)
    quit
```

各ラベルの説明を以下に示します。

^%ZSTART

このルーチンはまず QUIT コマンドを実行します。エントリ・ポイントの 1 つから実行を開始するのではなく、ルーチンとして呼び出す方が良好な結果を得ることができるためです。

このルーチンも、そのルーチン自体の名前、開始文字列および空文字列に対し、指定された定数 (マクロとして) を定義します。

SYSTEM^%ZSTART

このサブルーチンは、呼び出しルーチン名、エントリ・ポイント、呼び出された日付時刻で構成された文字列を作成します。その後、WriteConsole^%ZSSUtil を呼び出してオペレータのメッセージ・ログに配置します。

その後、表示する名前と値の組み合わせのリストを作成します。これを WriteLog^%ZSSUtil に渡し、ローカル・ログ・ファイルに配置します。その後、呼び出し元に戻ります。

LOGIN^%ZSTART、JOB^%ZSTART、および CALLIN^%ZSTART

これらのサブルーチンは、オペレータのメッセージ・ログに情報を置きません。その代わり、呼び出す際に識別するための簡単な項目のリストを作成し、WriteLog^%ZSSUtil を使用して記録します。

3.5.3 ^%ZSTOP

このルーチンは、実際に InterSystems IRIS に呼び出されるエントリ・ポイントを含み、^%ZSSUtil のサブルーチンを使用します。この例は、^%ZSTART の例と似ています。詳細は、前のセクションを参照してください。

ObjectScript

```
%ZSTOP ; User shutdown routine.

#define ME "ZSTOP"
#define EndSet "End"
#define Empty ""

    ; cannot be invoked directly
    quit

SYSTEM ; InterSystems IRIS stopping
    new EntryPoint

    set EntryPoint = "SYSTEM"
    ; record termination in the messages log
    do WriteConsole^%ZSSUtil((EntryPoint
        _ "^%"
        _ $$$ME
        _ " called @ "
        _ $ZDATETIME($HOROLOG, 3)))
    ; write the standard log information
    do Logit(EntryPoint, $$$ME)
    quit

LOGIN ; a user logs out of InterSystems IRIS
    new EntryPoint

    set EntryPoint = "LOGIN"
    do Logit(EntryPoint, $$$ME)
    quit

JOB ; JOB'd process exits.
    new EntryPoint

    set EntryPoint = "JOB"
    do Logit(EntryPoint, $$$ME)
    quit

CALLIN ; process exits via CALLIN interface.
    new EntryPoint

    set EntryPoint = "CALLIN"
    do Logit(EntryPoint, $$$ME)
    quit

Logit(entrypoint, caller) PRIVATE ;
    ; common logging for exits

    new items

    set items = $LISTBUILD($$$EndSet, $ZDATETIME($HOROLOG, 3))
    do WriteLog^%ZSSUtil(caller, entrypoint, items)
    quit
```

4

^%ZLANG ルーチンによる言語の拡張

^%ZLANG 機能を使用すると、ObjectScript などの言語に、カスタムのコマンド、関数、および特殊変数を追加することができます。拡張機能は標準機能と同じ方法で呼び出され、構文、演算子の優先順位などに関する同一の規則に従います。通常、^%ZLANG 機能の実行速度は標準の InterSystems IRIS 機能ほど速くありません。パフォーマンスが重要なルーチンをコーディングする際には、この点を考慮してください。

このような拡張機能を追加する手順は以下のとおりです。

1. 必要に応じて、以下の名前でルーチンを定義します。

ルーチン名	目的
^%ZLANGC00	カスタムの ObjectScript コマンドを定義します。
^%ZLANGF00	カスタムの ObjectScript 関数を定義します。
^%ZLANGV00	カスタムの ObjectScript 特殊変数を定義します。

2. これらのルーチンでは、パブリックのサブルーチンを以下のように定義します。
   - サブルーチンのラベルでは、定義するコマンド、関数、または特殊変数の名前を使用します。

     名前は英文字の Z で始まる必要があり、英字のみを含めることができます。ロケールでアルファベットとして定義すれば、Unicode 文字が使用できます。英文字はすべて大文字にする必要がありますが、実行の際は大文字と小文字の区別はありません。名前の最大長は 31 文字です。

     名前は既存のコマンド、関数、または特殊変数と同じにすることはできません (同一の場合、InterSystems IRIS では無視されます)。SQL 予約語は使用しないことを強くお勧めします。

     関数を定義する場合、関数に引数がない場合でも、ラベルには括弧を含める必要があります。

     特殊変数のラベルには括弧を含めることができます。
   - オプションで別のラベルを含めて、略称を定義します。その略称が InterSystems IRIS で使用されていないように注意してください。
   - サブルーチンを適切に定義します。詳細は、“メモ” セクションを参照してください。

3. 親ルーチンにある最初のコマンドでは QUIT を使用することをお勧めします。こうしておけば、ユーザがルーチンを直接呼び出しても、何も起こりません。

   また、ルーチンでコメントを上部に記載して、ルーチン自体の名前を示すようにすると便利です。

注釈　(正確ではありませんが) 慣習的に、ルーチンは名前の一部にキャレットがあるかのように参照されます。このドキュメントでは、この慣習に従います。

注意　他のサブルーチンの場合、パブリックのサブルーチンで呼び出されるものを含めて、それらのラベルは必ず、小文字のみか大文字と小文字を混在して記述します (先頭は Z にしない)。または、プライベートのサブルーチンとして実装します。

つまり、^%ZLANG ルーチンによって言語が拡張されるので、目的のサブルーチンのみが外部で利用できるようにすることが重要です。

4.1 メモ

コマンドは、ルーチンやプロシージャの DO のように処理されます。引数は、呼び出しのパラメータとして渡されます。

コードの結果にする場合を除いて、$TEST や $ZREFERENCE など、システム状態の値をコードで保持する必要があります (ただし、関数や特殊変数の場合、システムでは $TEST が自動的に保持されます)。

特殊変数の値を設定できます。変数のエントリ・ポイントは 1 つのみです。値を設定するか値を取得するかを判断するには、引数が指定されているかどうかをコードで調べる必要があります。以下はその例です。

ObjectScript

```
ZVAR(NewValue) public {
        if $DATA(NewValue) Set ^||MyVar=NewValue Quit
        Quit $GET(^||MyVar)
}
```

そうすると、以下に示すように、ユーザはこの変数を設定するか、または変数を取得できます。

```
USER>w $ZVAR

USER>s $ZVAR="xyz"

USER>w $ZVAR
xyz
```

コマンドまたは関数からエラー・コードを返すには、$SYSTEM.Process.ThrowError() を使用します。

4.2 例

例えば、カスタムの特殊変数を定義して ObjectScript で使用するには、以下のようにして、^%ZLANGV00 ルーチンを定義します。

ObjectScript

```
 ; implementation of ^%ZLANGV00
 ; custom special variables for ObjectScript
  QUIT

ZVERNUM        ; tag becomes name of a special variable
ZVE
  QUIT $PIECE($ZVERSION,"(Build")
```

次に、デモンストレーションとして、新規の変数はターミナルで以下のように使用できます。

```
USER>w $zvernum
InterSystems IRIS for Windows (x86-64) 2018.1
USER>w $zve
InterSystems IRIS for Windows (x86-64) 2018.1
```

別の例として、以下のように ^%ZLANGF00 ルーチンを定義するとします。

ObjectScript

```
 ; implementation of ^%ZLANGF00
 ; custom functions for ObjectScript
 QUIT

ZCUBE(input) public {
 Quit input*input*input
}
```

次に、デモンストレーションとして、新規の関数はターミナルで以下のように使用できます。

```
USER>w $zcube(2)
8
```

以下の例では ^%ZLANGC00 を示します。この例では、システム・ステータス・ユーティリティ ^%SS を実行するコマンドが作成されます。

ObjectScript

```
 ; %ZLANGC00
 ; custom commands for ObjectScript
  QUIT

ZSS       ; tag name of a command to check system status
  DO ^%SS
  QUIT
```

5

Windows クライアントからの InterSystems IRIS の制御

InterSystems IRIS には、InterSystems IRIS 構成を制御し、InterSystems IRIS プロセスを開始する Windows クライアント・プログラムのメカニズムが用意されています。これにより、標準 InterSystems IRIS ツールを使用せずに、正確な構成情報で InterSystems IRIS プロセスを自動的に開始するアプリケーションを配布できます。このツールにより、以下が可能になります。

- 指定された構成名の InterSystems IRIS ディレクトリ・パスとサービス名を検索
- InterSystems IRIS システムの状態を取得
- InterSystems IRIS 構成を直接、あるいは InterSystems IRIS コントロール・サービスから制御。これは、実行中の Windows バージョンに依存します。
- 適切な設定で InterSystems IRIS プロセスを開始

irisctl.dll を動的にロードし、その機能を使用することにより、これらのアクションを実行できます。

5.1 IRISctlGetDirs

指定された構成名に対して、構成・バイナリ・マネージャ、それぞれのディレクトリ・パスと、サービス名を検索します。

構文

```
IRISctlGetDirs(char *config, IRISCTL_DIR_INFO *dirinfo)
```

config	要求される構成名
dirinfo	ディレクトリ情報が格納される C 構造へのポインタ

返り値

エラー時に (char *0) 返します。

5.2 IRISctlConfigStatus

InterSystems IRIS 構成の状態を返します。

構文

```
IRISctlConfigStatus(char* config)
```

config	要求される構成名

返り値

以下の 0 から 4 までの値を返します。

0	構成は起動され、実行中です。
1	構成は、開始あるいは終了中です。
2	構成の開始あるいはシャット・ダウンがアボートされました。
3	構成はダウンしました。
4	エラー

5.3 IRISctlControl

Windows NT の InterSystems IRIS コントロール・サービスを経由して、あるいは Windows 95/98 で直接 InterSystems IRIS 構成を制御します。

構文

```
IRISctlControl(char *command, char *config)
```

command	以下のコマンドのいずれか 1 つを使用します。 ・ start — 構成を開始 ・ stop — 構成の無理のないシャット・ダウン ・ stopnoshut — ユーザ提供のシャット・ダウン・ルーチンを実行せずに、構成をシャット・ダウン ・ force — 構成の強制終了。UNIX® システムの irisforce と同じです。 ・ stopstart — 無理のない構成の停止と、すばやい再起動
config	要求される構成名

返り値

IRISCTL_SUCCESS	処理の成功
IRISCTL_ERROR	汎用エラー
IRISCTL_INVALID_COMMAND	無効なコマンド引数
IRISCTL_INVALID_CONFIGURATION	未定義の構成
IRISCTL_CONTROL_STU_ERROR	^STU の失敗

IRISctlGetLastError は、エラーの返り値の後ろに、エラー文字列情報へのポインタを返します。

5.4 IRISctlRun

指示された構成とネームスペースで InterSystems IRIS プロセスを開始し、指示された主要入出力デバイスを使用し、指示されたルーチンを呼び出します。

構文

```
IRISctlRun(char *config, char *routine, char *namespace, char *IOtype)
```

config	実行中の構成名です。
routine	開始するために要求されるルーチン名です。
namespace	要求されるネームスペース名です。
IOtype	入出力の処理方法です。以下の値を指定できます。 ・ terminal – プロセスによって、新しい InterSystems IRIS プログラマのターミナルが起動します。 ・ none – 入出力なし。プロセスは、NUL でバックグラウンドで実行されます。$Principal で使用されます。$Principal への書き込みは破棄されます。$Principal からの読み取りはエラーになります。

返り値

IRISCTL_SUCCESS	処理の成功
IRISCTL_ERROR	汎用エラー
IRISCTL_INVALID_COMMAND	無効なコマンド引数
IRISCTL_INVALID_CONFIGURATION	未定義の構成
IRISCTL_CONTROL_STU_ERROR	^STU の失敗

注釈　Windows NT では、指定の構成を実行する必要があります。構成が実行中かどうかが判別できない場合、IRISctlConfigStatus と IRISctlControl を使用してチェックし、必要な構成を開始してください。これにより InterSystems IRIS は、コントロール・サービスを使用せずに構成を開始しないようにします。

5.5 IRISctlRunIO

指示された構成とネームスペースで InterSystems IRIS プロセスを開始し、指示された主要入出力デバイス・タイプを使用し、指示されたルーチンと、入出力デバイスおよびエラー・デバイス用の追加の入出力仕様を呼び出します。

構文

```
IRISctlRunIO(
        char *config,
        char *routine,
        char *namespace,
        char *IOtype,
        HANDLE *hIO,
        char *cwd,
        char *options,
        HANDLE *child,
        DWORD *childPID))
```

config	実行中の構成名で、すべて大文字です。
routine	開始するために要求されるルーチン名です。
namespace	要求されるネームスペース名です。
IOtype	入出力の処理方法です。プロセスで TCP ソケットが使用されるため、値は TCP である必要があります。
hIO	3 つの配列は、InterSystems IRIS プロセスの標準入出力、エラー・デバイスとして使用するために処理します。
cwd	子プロセスの作業中のディレクトリ・パスです。ディレクトリ引数が 0 の場合、現在のプロセスの作業ディレクトリを使用します。
option	追加の irisdb.exe コマンド行オプションが、生成されたコマンド行に追加されます。例えば、大規模プロセスのメモリ・サイズ (-b 1024) を定義できます。
child	子プロセスへのハンドルが返される HANDLE タイプ変数へのポインタです。ハンドルの値が 0 の場合、子プロセスのハンドルは、この関数によってクローズされます。
childPID	生成された irisdb.exe プロセスの PID へのポインタです。この引数は、子の PID が要求されない場合、ゼロになります。

返り値

IRISCTL_SUCCESS	処理の成功
IRISCTL_ERROR	汎用エラー
IRISCTL_INVALID_COMMAND	無効なコマンド引数
IRISCTL_INVALID_CONFIGURATION	未定義の構成
IRISCTL_CONTROL_STU_ERROR	^STU の失敗

注釈　hIO 配列のハンドルは、必ず継承できる必要があります。DuplicateHandle を使用して、子プロセスによるハンドルの継承を可能にします。

Windows NT では、指定の構成を実行する必要があります。構成が実行中かどうかが判別できない場合、IRISctlConfigStatus と IRISctlControl を使用してチェックし、必要な構成を開始してください。これにより InterSystems IRIS は、コントロール・サービスを使用せずに構成を開始しないようにします。

6

グローバルの高速コピーのための ^GBLOCKCOPY の使用法

^GBLOCKCOPY は、データベース間でグローバルの高速コピーを行う InterSystems IRIS ルーチンです。動作モードには、インタラクティブとバッチの 2 種類があります。インタラクティブ・モードでは単一のプロセスを実行しますが、バッチ・モードでは複数のプロセスを並列で構成して実行できます。^GBLOCKCOPY には組み込みのモニタと、グローバル・コピーの複数の進行を記録するための複数のレポートが含まれます。システム障害が発生した場合、^GBLOCKCOPY は中断した後に再開します。

注釈　コピー中のデータベース・ブロックではロックも整合性もチェックされないため、^GBLOCKCOPY をグローバルのコピーに使用するのは、それらのグローバルがアクティブには変更されていない場合に限定する必要があります。Set や Kill の処理は、コピー先のグローバル、データベース、またはネームスペースにおいてだけでなく、コピー実行中のコピー元データベースにおいて他のグローバルに対して実行でき、コピーに影響を与えることもありませんが、別のデータベースやネームスペースにコピーされているコピー元グローバルで、Sets や Kills を行うと、コピー先のグローバルは予測できなくなります。

^GBLOCKCOPY がグローバルを新規のデータベースにコピーするとき、保護、ジャーナル属性、照合タイプ、および Keep 属性を含むソース・グローバルの同じプロパティを持つグローバルをそこに作成します。

注釈　SYS.Database.Copy() クラス・メソッドは、^GBLOCKCOPY と同様の機能を提供します。

6.1 ^GBLOCKCOPY の使用

^GBLOCKCOPY は、以下のように異なる複数の演算で使用されます。

- データベースから他のデータベース、またはネームスペースに 1 つ以上のグローバルをコピーします ー コピー先のデータベースまたはネームスペースにコピーするグローバルを 1 つ以上選択できます。グローバルがコピー先のデータベースに既に存在する場合、ソース・グローバルのデータは既存のデータに結合されます。
- 添え字レベル・マッピングを使用して、1 つのデータベースからのグローバルを複数のデータベースに分割します ーグローバルの添え字レベル・マッピング (SLM) でネームスペースを設定することで、データベースのグローバルを新規のネームスペースにコピーすることができます。これにより、SLM を構築するデータベース間でグローバルを分割することができます。
- 複数のデータベース内の添え字マップされたグローバルを、1 つのデータベースに移動します ー グローバル全体を含む新規のデータベースを作成します。次に、異なるすべての SLM データベースから新規のデータベースにコピーするバッチで、複数のコピーを設定します。

- データベースのコピーを作成します ー すべてのグローバルを別のディレクトリにコピーすることによって、データベースをそのディレクトリにコピーできます。
- ECP を通じて他のマシンにグローバルをコピーします ー ^GBLOCKCOPY は、ECP ネットワーク接続上の他のマシンへのグローバルのコピーをサポートします。リモート・マシンに対する ECP 接続、およびそれを示すネームスペース・マッピングを設定する必要があります。次に、“Copy from Database to Namespace” オプションを選択し、コピー先としてリモート・ネームスペースを選択します。
- データベースの未使用スペースを再要求します ー 大規模なグローバルがデータベース内で作成されて削除された場合、データベースには未使用の余分なスペースがあることがあります。データベースのすべてのグローバルを新規のデータベースにコピーし、古いデータベースを新しいもので置換すると、このスペースを削除できます。
- データベース内のポインタを再編成します ー データベースがブロックの分割により断片化された場合、そのデータベース内のデータを再編成すると高速なパフォーマンスを得ることができます。これには、データベース内のすべてのグローバルを新規のデータベースにコピーし、古いデータベースを新規のデータベースに置換します。
- グローバルの照合を変更します ー コピー対象の既存グローバルの照合を変更する場合、^GBLOCKCOPY を実行する前に、目的の既定の照合を指定して、コピー先データベースにグローバルを作成できます。
- Caché または従来のデータベースまたはネームスペースを InterSystems IRIS にインポートします ー CACHE.DAT ファイルまたは従来のデータベース・ファイルを IRIS.DAT データベースまたはネームスペースにインポートする場合、コピー元のディレクトリとして存在するディレクトリを選択するだけです。データベースは IRIS.DAT という名前に変更され、データをコピー先のデータベースやネームスペースにコピーすることができるようになります。

6.2 ^GBLOCKCOPY の実行

^GBLOCKCOPY を実行する前（またはアップグレードを実行する前）に、ご使用のデータベースのオペレーティング・システムの完全なバックアップを作成し、データベースに破損がないことを確実にするために整合性チェックを実行します。

注釈　^GBLOCKCOPY の実行を高速にするため、一時データやスクラッチ・データ、および不必要な古いデータはすべて削除します。

^GBLOCKCOPY の実行では、特定のデータベースにあるすべてのグローバルまたは一部のグローバルのみをコピーできます。このルーチンでは、コピーするグローバルの名前を指定することが要求されます。一部のグローバルのみを参照する場合、構文は次のようになります。

- glonam と入力し、コピーするグローバルとして “glonam” を選択します。
- glonam と入力し、選択しているグローバル “glonam” を選択解除します。
- glonam1-glonam2 と入力し、“glonam1” から “glonam2” までの範囲のグローバルを選択します。
- glonam1-glonam2 と入力し、“glonam1” から “glonam2” までの範囲のグローバルを選択解除します。

グローバルの選択で使用できるワイルドカード文字は次のとおりです。

- アンパサンド (&) は任意の 1 文字に一致します。
- 番号記号 (#) は 1 桁に一致します。
- 疑問符 (?) は任意の 1 文字に一致します。
- アスタリスク (*) は任意の個数の文字に一致します。

注釈　グローバルの範囲の指定ではワイルドカード文字を使用できません。

グローバルの一部を選択するときは以下の入力も使用できます。

- ? ー 情報入力を列挙します。
- ?L ー 利用可能なグローバルをすべて列挙し、選択済みのグローバルを番号記号 (#) で示します。
- ?S ー 現在選択されているグローバルを列挙します。
- ?H ー グローバル選択構文を列挙します。

^GBLOCKCOPY のバッチ機能を使用して、同時に実行する複数の操作を設定できます。バッチの構成時には、コピー・プロセスの総数と、ディレクトリごとのコピー・プロセスの最大数を入力することが求められます。グローバルごとに最大で 1 つのプロセスを実行できます。バッチ・モードでは、コピー処理の時間が最大グローバルのサイズに比例し、それにストレージの遅延と帯域幅も要因として関与します。操作バッチが実行されている間、Monitor、または Batch Report を使用して進行状況を監視することができます。

注釈　データベースを ^GBLOCKCOPY で処理している間は、データベースにアクセスしないようにしてください。^GBLOCKCOPY の実行中にアクセスすると、データベース操作の結果は予測不可能になります。同じシステム上で、^GBLOCKCOPY で処理しないデータベースは、安全に使用することができます。

7

スイッチの使用法

InterSystems IRIS スイッチは、インスタンス単位のフラグで、さまざまな目的に使用できます。特に、バックアップの実行中やクラッシュしたシステムの回復中に、各種システム・プロセスを抑制する際に便利です。^SWSET ルーチンを使用して、スイッチの値を直接操作できます。

背景

InterSystems IRIS のスイッチの起源はマシンとの物理的な接点にあり、かつてはコンピュータのオペレータ・コンソールの一部として使用されたり、マイクロコンピュータのフロント・パネルに装備されていました。オペレータは、これらのスイッチの 1 つを設定することで、そのマシンで実行中のプログラムに 1 ビットの情報を送信できました。InterSystems IRIS では“仮想マシン”が実装されるため、マシンに対するスイッチの概念も同様に抽象化されています。

今日、InterSystems IRIS におけるスイッチとは、InterSystems IRIS インスタンスの共有メモリまたは共通メモリの個々のビット設定であり、すべての InterSystems IRIS プロセスから見えます。一部のスイッチはユーザに対して設定されますが、その大半は InterSystems IRIS 自体のオペレーションに影響を与えます。

注釈　ユーザはスイッチを InterSystems IRIS インスタンスに対してローカルに使用する必要があります。InterSystems IRIS 自体には特定の設定の意味をクラスタの他のメンバに伝播する機能がありますが、これらの機能はインターシステムズの内部使用にのみ制限されています。ユーザ・スイッチの値は、他のシステムに送信することはできません。

7.1 現在定義されているスイッチ

スイッチは、すべて番号によって識別されます。InterSystems IRIS の起動時、スイッチはゼロ（オフ）に初期化されます。次のテーブルは、スイッチの番号とその動作を示しています。

スイッチ	意味/使用法
0 – 7	アプリケーション・プログラムで使用する目的で予約されています。
8	既存の InterSystems IRIS デーモンによるネットワーク要求に対する応答を抑制します。
9	ネットワーク・ログインを処理する新しいデーモンの作成を抑制します。
10	このスイッチを設定するプロセス以外によるすべてのグローバル・アクセスを抑制します。また、このプロセスを除き、ディスク入出力の原因となるルーチン・アクセスを抑制します。
11	このスイッチを設定するシステム・ジョブ以外によるすべてのグローバル・アクセスを抑制します。この設定はスイッチ 10 に優先し、システム用に予約されています。例えば、このスイッチは、コピー操作の前にシステムの動作を停止する目的でバックアップ・プロセスによって設定されます。

スイッチ	意味/使用法
12	InterSystems IRIS へのログインを抑制します。ユーザがログインを試みると、"Sign-on and JOB inhibited: Switch 12 is set" というメッセージが表示されます。
13	すべてのグローバル SET、KILL、および ZSAVE コマンドを抑制し、グローバルとルーチンに対して読み取りアクセスのみを許可します。
14	すべてのグローバルおよびすべてのルーチンに対するすべてのアクセスを抑制します。
15	通常はスイッチ 10、13、または 14 によってアクセスが抑制される場合でも、ピアからのネットワーク参照を許可します。
16	シャットダウン操作を調整する目的で、InterSystems IRIS によって内部的に使用されます。
17	クラスタでのジャーナル・フラッシュの完了待機を回避します。
18	ブロックに対するキューが非常に長くなる場合は、追加プロセスの休止を抑制します。
19	新規トランザクションの開始を抑制します。
20 – 31	未定義で、インターシステムズ用に予約されています。

注意　インターシステムズの担当者またはドキュメントに記載されている手順によって具体的に指示される場合を除き、アプリケーションで使用するスイッチは、アプリケーション・プログラム用に予約されているスイッチ (スイッチ 0-7) に限定する必要があります。

7.2 スイッチの操作法

^SWSET ルーチンを使用して、スイッチの値を直接操作できます。また、クラスタ・システム上でのジャーナル操作やシステム・バックアップを行う機能など、他の InterSystems IRIS 機能でも呼び出し元に代わってスイッチの値を設定します。

7.2.1 SWSET ルーチン

このルーチンは、ターミナル・セッションなどからスイッチの値をインタラクティブに設定する方法を提供します。

SWSET

パラメータ

なし

備考

この後の例に示す方法でこのルーチンを呼び出すと、スイッチ番号とそのスイッチに設定する値 (0 または 1) の入力を求められます。

例

以下の例は、SWSET の使用法を示しています。実行すると、

```
DO ^SWSET
```

以下のような内容が順次表示されます。

```
Set/Clear switch #:
Set/Clear switch #: 2
Set/Clear switch #: 2 to value (0 or 1):
Set/Clear switch #: 2 to value (0 or 1): 1
Set/Clear switch #: 2 to value (0 or 1): 1...done
```

7.2.2 %swstat^SWSET 関数

この関数は、スイッチの現在の設定を返します。

```
%swstat^SWSET(switch)
```

パラメータ

- switch ー スイッチの番号。

備考

考えられる返り値として以下の 3 つがあります。

- 0 ー スイッチが目的の値に設定されていないことを示します。
- 1 ー スイッチが新しい目的の値に適切に設定されていることを示します。
- -1 ー スイッチが無効な値 (0 または 1 以外の値) に設定されていることを示します。

例

以下の例は、スイッチ番号 1 の値を出力します。

```
Write $$%swstat^SWSET(1)
```

7.2.3 %swset^SWSET 関数

この関数は、スイッチを指定された値に設定します。

```
%swset^SWSET(switch, value)
```

パラメータ

- switch ー スイッチの番号。
- value ー 設定する値 (0 または 1)。

備考

switch が有効な番号で value が 0 または 1 の場合、この関数はスイッチを指定された値に設定し、次を返します。

- 0 ー スイッチはリセットされました (オフ)。
- 1 ー スイッチは設定されました (オン)。

それ以外の場合は、エラーが発生したことを示す値の -1 を返します。

例

以下の例は、スイッチ番号 1 の値をオフに設定します。

```
Write $$%swset^SWSET(1, 0)
```

7.3 失敗モード

InterSystems IRIS プロセスがシステム予約スイッチのいずれかを設定し、その処理を適切にクリーンアップせずに終了した場合、システムが制限オペレーティング・モードのままになる場合があります。例えば、あるプロセスでスイッチ 12 を設定した後に重大な障害 (あるいは単純な停止) が発生した場合、InterSystems IRIS は以降のユーザがログインできない状態になります。この問題が生じた場合は、インターシステムズのサポート窓口 (WRC) までお問い合わせください。

注釈　InterSystems IRIS では、スイッチ 10 に対してのみ自動リカバリが実装されています。プロセスがスイッチ 10 を設定後に停止した場合、InterSystems IRIS は自動的にこのスイッチをゼロにリセットします。

8

ルーチンによる InterSystems IRIS の管理に関する注意事項

管理ポータルによって、システム制御に使用する便利なブラウザベースのインタフェースが提供されます。ただし、ブラウザからシステムを管理できない場合にこのようなシナリオに対応するため、InterSystems IRIS では、同じタスクを実行するインタラクティブなルーチンを提供しています。ターミナルで実行する場合、これらのルーチンは、コマンド行インタフェースとして機能します。旧ドキュメントでは、このようなルーチンは、文字ベースのルーチンと呼ばれています。これらのルーチンについては、別の場所で詳細に説明されています。ここでは、これらすべてのルーチンに適用される追加情報を示します。

注意　同じルーチンの複数のインスタンスが、異なるシステム管理者（あるいは同じ管理者）によって同時に実行されるのを防止する方法はありません。こうした状況になった場合は、影響を受けるデータの一貫性に考慮し、各インスタンスの動作を調整して競合を回避して、目的を達成するのは管理者の責任となります。

重要　このドキュメントに説明されているルーチンを使用する場合、選択オプションに適するパラメータ値と InterSystems IRIS の動作について十分な運用知識が管理者にあることが前提となります。

ターミナルからこれらのルーチンを使用するには、ユーザが **%SYS** ネームスペースにいることと、最低でも **%Manager** ロールに属していることが必要です。ルーチンを開始するには、DO コマンドを使用します。例えば、^LEGACYNETWORK ルーチン（従来のネットワーク・ツールの構成をサポートするルーチン）を呼び出すには、次のコマンドを使用します。

ObjectScript

```
DO ^LEGACYNETWORK
```

開始時に、まずこれらの各オプションがオプションのリストと共に示されます。目的のオプションを選択するには、"Option?" プロンプトの後に対応する番号を入力します。ほとんどの場合、初期メニューを選択すると、さらにメニューが表示されるか、またはルーチンのタスク実行に必要な情報がすべて指定されるまで、必要な情報の入力が求められます。

以下の点に注意してください。

- 各オプションには、数字の接頭語があります。その番号を入力することで、オプションを選択します。オプション番号の形式は、すべてのルーチンで使用されています。
- すべてのオプション・リストに、メニューの現行レベルを終了して前のレベルに戻るための項目があります。または、"Option?" プロンプトに対して Enter キーを押して対応することもできます。この操作は [終了] オプションを選択したと同義に解釈されます。つまり、現行のセクションが終了され、1 つ "上位" レベルのオプションが表示されます。最上位レベルのオプションに対して Enter を押すと、ルーチンが終了します。

- 情報の入力を求めるプロンプトの多くには既定値があり、それは Enter キーを押すことで選択できます。使用可能な既定値がある場合は、次に示すように、プロンプト・メッセージと“=>”文字の間に表示されます。

  ```
  Unsuccessful login attempts before locking user? 5 =>
  ```

  この例では、既定値は 5 で、これはユーザが何回ログインに失敗するとそのユーザ名がロックされるかを表しています。

- 既定値が“Yes”または“No”のプロンプトでは、“yE”や“n”などの部分的に一致する応答も受け入れられます。この照合では、応答の大文字/小文字が無視されます。

- 既存のユーザ、ロール、サービスなどの設定変更が目的のオプションでは、それらの項目の既存値が既定として表示されます。Enter キーを押すと、その値が保存され、次のプロンプトに進みます。

- 一部のプロンプトでは、ユーザ名などの項目を照合するときに使用するパターンの入力が求められます。通常、既定のパターンは、すべての項目に一致する“*”です。このパターンでは、DOS における照合と同じように、アスタリスクが任意の文字シーケンスに一致します。パターンは、それぞれが固有のパターンとして解釈されるコンマ区切りのパターン・リストで構成される場合もあります。対象の項目がリスト内のいずれかのパターンに一致すると、その項目は選択されたものとして処理されます。

9
プロセス管理

InterSystems IRIS のプログラムとアプリケーションはすべて、プロセスによって実行されます。プロセスはシステムが生成する整数値のプロセス ID (pid) によって識別されます。

プロセスはフォアグラウンド (対話型) プロセスまたはバックグラウンド (非対話型) プロセスのいずれかになります。バックグラウンド・プロセスは JOB コマンドを使用して ObjectScript で開始されます。JOB コマンドを発行するプロセスは親プロセスと呼ばれ、開始されたバックグラウンド・プロセスは子プロセスと呼ばれます。バックグラウンド・プロセスは「ジョブ起動プロセス」または「生成されたプロセス」と呼ばれることもあります。

プロセスはロックを使用して共有リソースの保持と解放を行います。ObjectScript では、プロセスは LOCK コマンドを使用してロックの取得と解放を行います。現在のロックを表示するには、"ObjectScript の使用法" の "ロック管理" の章で説明されているシステム全体のロック・テーブルを参照してください。

この章では、プロセスのバッチ・モードとプロセスの優先度について説明します。プロセス管理のその他の側面については、%SYS.ProcessQuery クラスを参照してください。

注釈　InterSystems IRIS では、プロセス管理のすべての側面に対して最適な既定が用意されています。プロセスのこれらの既定の変更は、特定の特別な状況に限って慎重に行ってください。

9.1 バッチ・モード

プロセスは、インタラクティブ・モードと呼ばれることもある既定モード (これはユーザ対話とは無関係です) またはバッチ・モードのいずれかで実行されます。バッチ・モードは、適切なツールが利用できない特殊な状況でのみ使用してください。

データベースの広範囲の部分にアクセスするプロセスは、システムで実行されている他の (バッチでない) プロセスへの影響を限定するために、バッチ・モード・プロセスに設定される場合があります。特に、バッチ・モード・プロセスは、読み込みや変更対象のデータベース・ブロックによってデータベース・キャッシュを圧迫しないようになっています。例えば、データ圧縮ユーティリティはバッチ・モードで適切に実行される場合があります。

バッチ・モード・プロセスはジョブ起動プロセスと同じではありません。ジョブ起動プロセスは JOB コマンドを使用して生成されます。ジョブ起動プロセスはバックグラウンドで実行される非対話型のプロセスです。

%SYSTEM.Process.BatchFlag() メソッドを使用すると、現在のプロセスをバッチ・モードで実行できます。BatchFlag() はブーリアン引数を取ります。0=ダイレクト・モード (既定)、1=バッチ・モードです。引数が指定されていない BatchFlag() は、現在のモード設定を返します。

9.2 優先度

プロセスの優先度は、複数の並行プロセスに CPU リソースの競合が発生する仕組みを決定します。優先度が高いプロセスは CPU 時間に優先的にアクセスします。

優先度は正数値です。正数値の範囲はプラットフォームに依存します。Windows の場合、優先度の範囲は 1 ～ 15、通常=8 です。UNIX® の場合、優先度の範囲は -20 ～ 20、通常=0 です。一般的に、低、通常、高の 3 つの正数値のみが使用されます。通常優先度がユーザ・プロセスの既定値です。通常優先度はロード・バランシングを使用して CPU の使用率を調整します。

プロセスの現在の優先度は、**%SYS.ProcessQuery** クラスの Priority プロパティを使用して決定できます。

注釈　　プロセスの優先度の変更はほぼ不要です。変更するとシステムの安定性を損なう可能性があります。

9.2.1 SetPrio() メソッド

%SYSTEM.Util.SetPrio() メソッドを使用して、現在のプロセスまたは pid で識別される別のプロセスの優先度を変更できます。正または負の整数を指定して、現在の優先度をその分だけ増減します。Windows では、最小優先度は 1 です。優先度を 1 より小さい値にしようとしても、優先度は 1 に設定されます。通常優先度は 8 です。最大優先度は 15 です。SetPrio() は新しい優先度のレベル設定値を返します。(15 より大きい値を指定しようとすると、優先度は 1 にリセットされるか、その他の予期しない結果を招く可能性があります。15 を超える SetPrio() の戻り値は実際の優先度設定を反映していません。)

9.2.2 ^%PRIO ユーティリティ

古い ^%PRIO ユーティリティを使用して現在のプロセスの優先度を決定し、優先度を低、通常、高のいずれかの値に設定できます。

^%PRIO を使用して呼び出しや変更を行うには、**%DB_IRISSYS** リソースに書き込み権限でアクセスできる必要があります。

現在の優先度を決定するには、`DO ^%PRIO` コマンドを発行します。後続の引数のない WRITE コマンドは、現在の優先度を `%PRIO=8` のように返します。

現在のプロセスの優先度を設定するには、`DO LOW^%PRIO`、`DO NORMAL^%PRIO`、または`DO HIGH^%PRIO` のいずれかのコマンドを発行します (コマンドは大文字/小文字を区別します)。設定済みの優先度の戻り値 (Windows の場合) は低 =4、通常 =8、高 =13 です。

%SYSTEM.Process.BatchFlag() は、現在のプロセスの優先度を変更します。ただし、プロセスがバッチ・モード (1) の場合、上記のコマンドを使用して現在のプロセスの優先度を上げると、プロセスのモードはバッチ・モード (1) からダイレクト・モード (0) に変わります。

バッチ・モードと現在のプロセスの優先度の両方を設定するには、DO SET^%PRIO(priority,mode) を指定します。priority,mode は大文字/小文字が区別される文字列で、priority にはLOW、NORMAL、またはHIGHのいずれか、mode には、BATCH または NOBATCH のいずれかを指定します。コンマ区切りオプションのいずれか一方、または両方を指定できます。例：`DO SET^%PRIO("LOW,BATCH")`。このコマンドにより、バッチ・モード・プロセスのモード・ステータスに影響を与えることなくプロセスの優先度を変更できます。

9.2.3 ジョブ起動プロセスの優先度

JOB コマンドにはジョブ起動 (バックグラウンド) プロセスの優先度を設定するオプションが用意されています。指定しない場合、子プロセスはシステム定義のジョブ優先度変更子によって調整された親プロセスのベース優先度を使用しま

す。例えば、Windows の場合、通常優先度 (priority=8) の親プロセスは、ジョブ優先度変更子により、priority=7 の子プロセスを開始します。優先度が高のジョブ起動プロセスは、対話型プロセスと対等に CPU リソースを取り合います。

10

cvendian を使用したバイト・オーダー変換

この章では、ビッグ・エンディアン・プラットフォームとリトル・エンディアン・プラットフォームの間で移行するために、インターシステムズ・データベースのバイト・オーダーを変換するユーティリティについて説明します。また、指定されたデータベースのバイト・オーダーに関して報告するためのオプションも提供しています。

10.1 cvendian の概要

インターシステムズは、インターシステムズ・データベースのバイト・オーダーをビッグ・エンディアン（最上位のバイトが先頭）からリトル・エンディアン（最下位のバイトが先頭）、またはその逆に変換するユーティリティを提供しています。これは、convert endian を略して cvendian と呼ばれています。2 つのタイプのプラットフォーム間でデータベースを移動する際に役立ちます。また、指定されたデータベースのバイト・オーダーに関して報告するためのオプションも提供しています。

サポート対象プラットフォームのエンディアンについては、このリリース用のオンライン・ドキュメント″インターシステムズのサポート対象プラットフォーム″ の “プラットフォームのエンディアン” を参照してください。

重要　　このユーティリティは、マウント済みデータベースには使用できません。

10.1.1 ユーティリティの場所

cvendian ユーティリティは、ファイル install-dir¥Bin¥cvendian.exe です。

10.1.2 変換プロセス

cvendian は、変換するファイルを持つシステム上、または変換するファイルを使用する予定であるシステムのいずれかで実行可能です。

例えば、データベースをリトル・エンディアン・システムからビッグ・エンディアン・システムに変換するには、リトル・エンディアン・システムで変換を実行した後データベースをビッグ・エンディアン・システムに転送するか、ファイルを最初に転送した後変換できます。

注釈
- cvendian ユーティリティでは、簡潔なバッファ型 I/O を使用して、ターゲット・ファイルを読み取り、また書き込みます。最適なパフォーマンスを得るには、オペレーティング・システム (OS) のファイル・システム・キャッシュが無効でないことを確認します。
- バックアップ・ファイルやジャーナル・ファイルに対して、cvendian ユーティリティは使用できません。同じエンディアンのプラットフォームでデータベースをリストアし、リストアしたデータベースを別のエンディアン・プラットフォームに移動してから、cvendian ユーティリティを使用してデータベースを変換する必要があります。

データベースを変換するプロセスは、以下のとおりです。

1. ユーティリティはソース・ファイルを変換済みのファイルと置換するため、使用中のデータベース・ファイルのコピーを作成します。
2. “ユーティリティの構文”のセクションで説明する構文を使用して cvendian を実行します。

10.1.3 ユーティリティの構文

cvendian エンディアン・ユーティリティを使用すると、希望するバイト・オーダーを指定するか、変換せずに現在のバイト・オーダーを報告できます。以下の構文を使用します。

```
cvendian [-option] file
```

option 引数は以下のいずれかになります。

- `-big` – データベースをビッグ・エンディアンに変換
- `-little` – データベースをリトル・エンディアンに変換
- `-report` – データベースのバイト・オーダーを報告

オプションを短縮して頭文字で表すこともできます。これが変換要求 (-big または -little) であり、データベースのバイト・オーダーが既に指定したものになっている場合、警告メッセージが表示され、処理は停止します。

option 引数を指定しない場合、ユーティリティによって既存のバイト・オーダーから他のバイト・オーダーにデータベースが変換されます。ただし、option 引数を使用することをお勧めします。

file 引数は変換するファイルで、完全なパス名を含むことができます。

ユーティリティは、以下のアクションを実行します。

- データベースのバイト・オーダーを自動的に検出します。
- エンディアン情報と他の情報を表示します。
- 変換を実行します。
- 成功、または失敗を示すメッセージを表示します。

ここでは、Windows Server 2016 から IBM AIX® for Power System-64 で使用するためにデータベースを変換すると想定します。つまり、リトル・エンディアン (Intel) からビッグ・エンディアン (POWER) に変換する必要があります。AIX システムへファイルを移動する前に Windows システムで cvendian を実行した場合の出力は以下のようになります。

```
C:\IrisSys\Bin>cvendian -big c:\temp\powerdb\iris.dat

This database is little-endian.
This database has a block size of 8192 bytes.

This database has 1 volume and 1 map.
The last block in the primary volume is 18176.

Original manager directory is c:\temp\powerdb\

No extension volumes.

Done converting c:\temp\powerdb\iris.dat to big-endian

C:\IrisSys\Bin>
```

次に、変換済みのデータベース・ファイルを AIX システムに移動できます。