$ZF コールアウト・インタフェースの使用法

Version 2024.1
2024-06-03

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

$ZF コールアウト・インタフェースの使用法

InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

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

目次

1

InterSystems $ZF コールアウト・インタフェース

このドキュメントで取り上げる内容の詳細なリストは、"目次" を参照してください。

InterSystems $ZF コールアウト・インタフェースは、外部システム呼び出しと関数呼び出しを InterSystems IRIS に統合するいくつかの方法を提供する ObjectScript 関数セットです。

- $ZF(-100) 関数は、コマンド行インタフェースから行うのと同じように、オペレーティング・システム・コマンドを呼び出し、外部プログラムを実行する簡単な方法を提供します。
- 他の **$ZF** 関数は、ObjectScript アプリケーションが頻繁に使用する C 関数にアクセスできるようにするさまざまなオプション (後述) を提供します。

注釈　$ZF コールアウト・インタフェースは元来、ObjectScript アプリケーションから C 関数ライブラリにアクセスできるようにするために設計された比較的古いインターシステムズのテクノロジです。Java、.NET、Python、Node.js などの外部言語に埋め込み言語からアクセスする場合は、InterSystems 外部サーバにより、もっと簡単で強力な方法が提供されています。

$ZF コールアウト・インタフェースは、簡単な OS コマンド行呼び出しの作成から、数百もの関数にアクセスできるようにする関数ライブラリの作成まで、さまざまな目的で使用できます。以下のオプションを使用できます。

外部プログラムを起動する呼び出しを含む、オペレーティング・システムの呼び出しへのアクセス

- $ZF(-100) を使用したプログラムまたはシステム・コマンドの実行

  コマンド行インタフェースから行うのと同じように、システム・コマンドを呼び出し、外部プログラムを実行します。この簡単な関数は、コールアウト・ライブラリを記述する必要がありません。

コールアウト・ライブラリのアクセス。コールアウト・ライブラリは、$ZF コールアウト・インタフェースへのフックを含む、ユーザが記述する共有ライブラリ (DLL または SO ファイル) であり、多様な $ZF 関数が実行時にロードして、その関数を呼び出すことができるようにします。コールアウト・ライブラリは通常 C 言語で記述されますが、互換性のある呼び出し規則を持つ言語も使用できます ("互換性のある言語とコンパイラ" を参照)。各 **$ZF** 関数は、主に、ライブラリの識別方法とメモリへのロード方法が異なります。

- $ZF() を使用した iriszf コールアウト・ライブラリへのアクセス

  iriszf と呼ばれる特別な共有メモリを作成します。このライブラリが利用可能な場合、事前にライブラリをロードしたり、ライブラリ名を指定したりしなくても、その関数にアクセスできます。

- $ZF(-3) を使用した単純なライブラリ関数呼び出し

  ライブラリをロードして、ライブラリ・ファイル・パスと関数名を指定することによって関数を呼び出します。使用は簡単ですが、仮想メモリ内で使用できるのは一度に 1 つのライブラリのみです。その他のインタフェースとは異なり、ライブラリ関数を呼び出す前に初期化を行う必要がありません。

- $ZF(-5) を使用したシステム ID によるライブラリへのアクセス

一度に複数のライブラリを効率的に維持し、アクセスするために使用できるインタフェースを作成します。同時に複数のライブラリをロードでき、それぞれに必要な処理オーバーヘッドは、$ZF(-3) よりはるかに小さくて済みます。

- ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用

  コールアウト・ライブラリの大規模なセットを処理するための最も効率的なインタフェースを作成します。ライブラリには、グローバルに定義されたインデックス・テーブルを介してアクセスします。インデックスは、InterSystems IRIS のインスタンス内のすべてのプロセスで利用でき、同時に複数のライブラリをメモリに配置できます。

$ZF 関数の概要については “$ZF インタフェースの概要”、すべての $ZF 関数の説明、使用情報、および詳細や例へのリンクは “InterSystems コールアウトのクイック・リファレンス” も参照してください。

2

$ZF コールアウト関数の概要

InterSystems IRIS® $ZF システム関数は、関連する関数スイートのコンテナです。$ZF スイート内のほとんどの関数は、関数呼び出しの最初の引数によって識別されます。これは -100 または -3 ～ -6 の負の数字です。例えば、オペレーティング・システム・コマンドを呼び出す関数は $ZF(-100,<oscommand>) の形式を持ちます。ここで、<oscommand> は、実行されるコマンドを含む文字列です。この関数について述べる場合には、$ZF(-100) と示されます。同様に、その他の関数は、実際の関数呼び出しの最初のパラメータのみを使用して $ZF(-3) ～ $ZF(-6) と示されます。$ZF() 関数は、負の数のパラメータを使用せずに呼び出すこともできます。その場合は、iriszf という名前の特殊なコールアウト・ライブラリから関数が呼び出されます。

注釈 **コールアウト・ライブラリ**

共有ライブラリは、動的にリンクされたファイル (Windows の DLL ファイル、または UNIX® および関連オペレーティング・システムの SO ファイル) です。コールアウト・ライブラリは、$ZF コールアウト・インタフェースへのフックを含む共有ライブラリであり、多様な $ZF 関数が実行時にロードして、ライブラリ関数を呼び出すことができるようにします。コールアウト・ライブラリの記述方法の詳細は、“InterSystems コールアウト・ライブラリの作成” を参照してください。

$ZF 関数スイートには、以下のインタフェースがあります。

$ZF() 関数 (負の数の引数なし)

メインの $ZF() 関数は、iriszf という名前の特殊なコールアウト・ライブラリから関数への直接アクセスを提供します。このカスタム・ライブラリを定義およびコンパイルすると、関数名と引数を指定するだけで (例えば $ZF("myFunction",arg1))、その関数を呼び出すことができます。$ZF(-3)、$ZF(-5)、または $ZF(-6) とは異なり、ライブラリをロードしたり、ライブラリ識別子を指定したりする必要はありません。

詳細は、″$ZF() を使用した iriszf コールアウト・ライブラリへのアクセス″ を参照してください。

$ZF(-100) 関数

$ZF(-100) 関数は、シェル・コマンドおよびオペレーティング・システム・サービスの呼び出しの実行に使用されます。これは、コールアウト・ライブラリへのアクセスには使用されず、前もって設定せずに呼び出すことができます。

詳細は、“$ZF(-100) を使用したプログラムまたはシステム・コマンドの実行” を参照してください。

$ZF(-3) 関数

$ZF(-3) 関数は、コールアウト・ライブラリをロードして単一の文でライブラリ関数を呼び出す簡単な方法です。ライブラリとその関数の両方が名前で指定され、ライブラリは異なるライブラリへの呼び出しによって置き換えられるまでメモリ内に残ります。

詳細は、“単純なライブラリ関数呼び出しでの $ZF(-3) の使用” を参照してください。

$ZF(-4) 関数

$ZF(-4) 関数は、$ZF(-5) および $ZF(-6) に一連のサービスを提供します。これは、$ZF(-4,1) から $ZF(-4,8) までの最初の 2 つのパラメータによって識別される 8 つのユーティリティ関数のコンテナです。$ZF(-5) 関数インタフェースは、関数 $ZF(-4,1) ～ $ZF(-4,3) を使用し、$ZF(-6) 関数インタフェースは、関数 $ZF(-4,5) ～ $ZF(-4,8) を使用します。詳細は、以下の説明を参照してください。

$ZF(-5) 関数インタフェース

$ZF(-5) 関数およびそのユーティリティ関数を使用すると、複数のライブラリを効率的に処理できます。ライブラリおよびその関数の両方がシステム定義の ID 値によって識別されます。複数のライブラリを同時に仮想メモリ内に入れることができます。以下の $ZF(-4) 関数はライブラリのロードやアンロード、およびライブラリや関数 ID 値の取得に使用されます。

- $ZF(-4,1) は、名前によって指定されるライブラリをロードし、ライブラリ ID を返します。
- $ZF(-4,2) は、ライブラリをアンロードします。
- $ZF(-4,3) は、指定されたライブラリ ID および関数名の関数 ID を返します。

詳細は、“システム ID によるライブラリへのアクセスでの $ZF(-5) の使用”を参照してください。

$ZF(-6) 関数インタフェース

$ZF(-6) 関数およびそのユーティリティ関数は、ハードコード化されたライブラリ名を必要としないコールアウト・アプリケーションを記述する方法を提供します。代わりに、実際のライブラリ・ファイル名は別のインデックス・テーブルに含まれます。ここでは、各ライブラリは一意のユーザ定義インデックス番号に関連付けられています。インデックス・テーブルが定義されると、InterSystems IRIS のインスタンス内のすべてのプロセスで利用可能になります。コールアウト・アプリケーションは、ライブラリをインデックス番号で識別し、インデックス・テーブルを読み取ることによってロードします。複数のライブラリを同時にメモリ内に入れることができます。以下の関数は、インデックスの管理とライブラリのロードまたはアンロードに使用されます。

- $ZF(-6) は、ライブラリ関数を呼び出し、ライブラリがまだメモリ内に無い場合には、ロードします。
- $ZF(-4,4) は、ライブラリをアンロードします。
- $ZF(-4,5) および $ZF(-4,6) は、InterSystems IRIS のインスタンス内のすべてのプロセスでアクセス可能なシステム・インデックス・テーブルを作成および維持するために使用されます。
- $ZF(-4,7) および $ZF(-4,8) は、単一プロセス内のシステム・インデックスをオーバーライドするために使用できるプロセス・インデックス・テーブルを作成および維持するために使用されます。

詳細は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用”を参照してください。

すべての $ZF 関数の完全なリスト、それらの使用方法に関する情報、および詳細な情報および例へのリンクについては、“InterSystems コールアウトのクイック・リファレンス”を参照してください。

3

$ZF(-100) を使用したプログラムまたはシステム・コマンドの実行

$ZF(-100) 関数は、InterSystems IRIS® プロセスによるホスト・オペレーティング・システムの実行可能プログラムまたはコマンドの呼び出しを許可します。これは、特別なコールアウト共有ライブラリなしで使用できる唯一の $ZF 関数です (“InterSystems コールアウト・ライブラリの作成” を参照してください)。この章で説明する項目は以下のとおりです。

- はじめに ー $ZF(-100) の構文および機能の概要。
- プログラムの実行ープログラムはオプションで非同期に実行することもオペレーティング・システム・シェル内で実行することもできます。
- コマンドの記録と出力のリダイレクトーオプションの設定を使用して、コマンドを記録したり、入出力をリダイレクトしたりできます。
- %System_Callout:USE 権限の追加 ー この権限は $ZF(-100) を使用するために必要です。

″ObjectScript リファレンス″ の ″$ZF(-100) (ObjectScript)″ も参照してください。

注釈　$ZF(-100) は、非推奨の関数 $ZF(-1) および $ZF(-2) に代わるもので、すべてのケースにおいてこちらを使用することをお勧めします。

3.1 はじめに

$ZF(-100) は、コマンド行インタフェースの機能と同様の機能を提供します。これを使用すると、ホスト・オペレーティング・システムの実行可能プログラムまたはコマンドを呼び出すことができます。この関数の構文は次のとおりです。

```
status = $ZF(-100, keywords, command, arguments )
```

最初の引数はリテラル `-100` である必要があります。その他の 3 つの引数には、以下の情報を指定します。

- keywords ー さまざまなオプションを指定するキーワードが含まれる文字列。例えば、文字列 `"/ASYNC/LOGCMD"` は、プログラムを非同期に実行し、コマンド行をログ・ファイルに書き込む必要があることを指定します。
- command ー 呼び出すプログラムまたはシステム・コマンドを指定する文字列。実行可能プログラムのフル・パスを指定しないと、オペレーティング・システムでは標準の検索パス・ルールが適用されます。
- arguments ー コマンドの引数は一連のコンマ区切り式として指定されます (以下の例で示しています)。

$ZF(-100) 関数は、オペレーティング・システムおよび呼び出されたプログラムによって判断された終了状態コードを返します。

以下の例では、3 つの文字列が echo コマンドに渡された後、状態コードが表示されます。この例ではキーワードは使用されていないので、keywords 引数は空の文字列になっています。最後のコマンド引数は、引用符付き文字列を指定します (標準の ObjectScript 文字列ルールに従っています)。

```
USER>set status = $ZF(-100,"","echo","hello","world","""goodbye now""")
hello world "goodbye now"

USER>write status
0
```

以下のセクションでは、さらに、さまざまな $ZF(-100) オプションの例を示しています。キーワードおよびその他のオプションの概要は、"コールアウトのクイック・リファレンス" の章の "$ZF(-100) : プログラムまたはシステム・コマンドの実行" を参照してください。

3.2 プログラムの実行

$ZF(-100) を使用すると、同期的または非同期的に、オペレーティング・システム・シェルを呼び出してまたは呼び出さずに、プログラムまたはコマンドを実行できます。既定では、シェルを呼び出さずに同期的に実行されます。既定の実行は、関数呼び出しでオプションのキーワードを指定することによってオーバーライドできます。

以下のキーワードを使用して、プログラム実行を制御できます。

- /ASYNC — プログラムを非同期で実行する必要があることを示します。これにより $ZF(-100) の呼び出しから、プログラムの完了を待たずに値が返されます。
- /SHELL — プログラムをオペレーティング・システム・シェル内で実行する必要があることを示します。

前のセクションで触れたように、これらのオプションのいずれも使用しない場合は、keyword パラメータに空の文字列を指定できます。この例では、エラー・コード 1 が生成されるように、存在しないファイルを意図的にリストしようとしています。

```
USER>set status = $ZF(-100,"", "ls","*.scala")
ls: cannot access *.scala: No such file or directory

USER>write status
1
```

同じコマンドを非同期で実行すると、エラー・コードが返されるため、出力は表示されず、status は未定義になります。

```
USER>kill status
USER>set status = $ZF(-100,"/ASYNC", "ls","*.scala")
USER>write status
WRITE status
^
<UNDEFINED> *status
```

このような状況でエラー出力をリダイレクトする方法は、次のセクションの "入出力リダイレクトの使用法" を参照してください。

3.3 コマンドの記録と出力のリダイレクト

以下のキーワードによって、ログへの記録と入出力リダイレクトが制御されます。

- /LOGCMD – プログラム・コマンドおよび引数がメッセージ・ログに送信されるようにします。
- /STDIN、/STDOUT、および /STDERR – これらを使用して、呼び出されたプログラムの標準の入力、標準の出力、および標準のエラーをリダイレクトします。これらのキーワードの後ろには、ファイルの指定を付ける必要があります(以下の“入出力リダイレクトの使用法”を参照してください)。

3.3.1 コマンド引数のログへの記録

/LOGCMD キーワードによって、コマンド引数および終了状態コードがメッセージ・ログ (<install-dir>¥mgr¥messages.log) に記録されるようになります。これは主に、$ZF(-100) に渡された式が実際にどのように評価されたかを確認するためのデバッグ・ツールになることを目的としています。

ほとんどの場合、コマンドおよびその引数は 1 行に記録され、次の行に戻り値が記録されます。例えば、Windows では set status=$ZF(-100,"/LOGCMD","echo","hello","world") によって以下のログ・エントリが生成されます。

```
03/28/18-11:49:51:898 (26171) 0 $ZF(-100) cmd=echo "hello" "world"
03/28/18-11:49:51:905 (26171) 0 $ZF(-100) ret=0
```

ただし、UNIX® では、/SHELL が指定されていない場合、以下のように値は 1 行に 1 つずつ記録されます。

```
03/28/18-12:09:22:243 (26171) 0 $ZF(-100) argv[0]=echo
03/28/18-12:09:22:500 (26171) 0 $ZF(-100) argv[1]=hello
03/28/18-12:09:22:559 (26171) 0 $ZF(-100) argv[2]=world
03/28/18-12:09:22:963 (26171) 0 $ZF(-100) ret=0
```

いずれの場合でも、引数はプログラムで受け取られるとおりに記録されます。

3.3.2 入出力リダイレクトの使用法

以下のキーワードとファイル指定子は、入出力リダイレクトを制御します。

- /STDIN=input-file
- /STDOUT=output-file または /STDOUT+=output-file
- /STDERR=error-file または /STDERR+=error-file

入出力リダイレクト・キーワードの後ろには、演算子 (= または +=) と、ファイル名またはファイル・パスが続きます。演算子の前後にスペースを入れることができます。標準の入力は、既存のファイルを指す必要があります。標準の出力および標準のエラー・ファイルは、存在しない場合は作成され、既に存在する場合は切り捨てられます。ファイルを作成または切り捨てるには = 演算子を使用し、既存のファイルに追加を行うには += 演算子を使用します。標準のエラーと標準の出力が同一のファイルに送信されるようにするには、両方のキーワードに同じファイルを指定します。

次の例の最初の行では echo コマンドからファイル temp.txt に標準の出力がリダイレクトされ、2 つ目の行に結果のファイル・コンテンツが表示されています。

```
USER>set status = $ZF(-100,"/STDOUT=""temp.txt""","echo","-e","three\ntwo\none\nblastoff")
USER>set status = $ZF(-100,"","cat","temp.txt")
three
two
one
blastoff
```

次の例では、ファイル temp.txt を標準の入力にリダイレクトすることによって、このファイルの 2 行を別の方法で表示しています。tail コマンドが入力を受け入れ、最後の 2 行を表示します。

```
USER>set status=$ZF(-100,"/STDIN=""temp.txt""","tail","-n2")
one
blastoff
```

次の最後の例では、標準のエラーが temp.txt にリダイレクトされ、存在しないファイルの表示を試行しています。ここでは、コマンドを非同期で実行するために /ASYNC キーワードも使用されており、$ZF(-100) 呼び出しが値を返してから、エラー・メッセージを表示できるようになっています。ここでも 2 つ目の行 (前の例とまったく同じ) にファイルの最後の 2 行が表示されており、リダイレクトされたエラー・メッセージが含まれています。

```
USER>set status = $ZF(-100,"/ASYNC /STDERR+=""temp.txt""","cat","nosuch.file")

USER>set status=$ZF(-100,"/STDIN=""temp.txt""","tail","-n2")
blastoff
cat: nosuch.file: No such file or directory
```

3.4 %System_Callout:USE 特権の追加

$ZF(-100) は、**%System_Callout:USE** 権限を必要とします。インターシステムズのセキュリティ設定が最小よりも高い場合は、この権限が無効になっていることがあります。以下の手順では、**%Developer** ロールでこの権限を有効にする方法を示します。

%Developer ロールでの %System_Callout:USE の有効化

1. 管理ポータルを開いて、**[システム管理]** > **[セキュリティ]** > **[ロール]** に移動します。
2. [ロール] ページで、[名前] 列の **[%Developer]** をクリックします。
3. [ロール %Developer の定義編集] ページの [General] タブで、**%System_Callout** 権限を見つけて、**[ ]** をクリックします。
4. [リソースの許可を編集] ダイアログで、**[ ]** チェック・ボックスにチェックが付いていない場合はチェックを付け、**[OK]** をクリックします。

%Developer ロールは InterSystems IRIS のインストール時に必ず作成されますが、一部のユーザはこのロールを使用できないように管理者が設定している場合があります。場合によっては、$ZF(-100) を使用可能にするが、その他の権限を付与しないロールをユーザに提供することが望ましいこともあります。以下の手順で、**%System_CallOut:USE** 権限のみを付与する新規ロールを作成します。

新規ロールでの %System_Callout:USE の有効化

1. 管理ポータルを開いて、**[システム管理]** > **[セキュリティ]** > **[ロール]** に移動します。
2. [ロール] ページで、**[ ]** ボタンをクリックして、[ロール編集] ページを開きます。
3. 以下のように名前と説明を入力します。
   - 名前: **UseCallout**
   - 説明: **%System_CallOut**

   [ ] をクリックすると、フォームに **[ ]** ボタンが表示されます。
4. **[ ]** ボタンをクリックすると、リソースのスクロール・リストが表示されます。リストから **%System_CallOut** を選択して **[ ]** をクリックします。[ロール編集] フォームの **[ ]** をクリックします。
5. [ロール] ページのロール定義のリストに新しい **UseCallout** ロールが表示されます。

4

InterSystems コールアウト・ライブラリの作成

InterSystems コールアウト・ライブラリは、カスタム・コールアウト関数、および InterSystems IRIS がそれらを使用できるようにする使用可能化コードを含む共有ライブラリです。この章では、コールアウト・ライブラリを作成して実行時にアクセスする方法を説明します。

以下の項目について説明します。

- コールアウト・ライブラリの概要 ー コールアウト・ライブラリの作成およびアクセス方法を説明します。
- ZFEntry リンク・オプション ー 関数引数が渡される方法を決定するリンク・オプションの詳細を説明します。
- 互換性のある言語とコンパイラ ー C 以外の言語でコールアウト・ライブラリを作成する方法を説明します。
- コールアウト・ライブラリ Runup 関数と Rundown 関数 ー コールアウト・ライブラリがロードまたはアンロードされるときに自動的に実行されるように設定できる 2 つのオプション関数について説明します。
- トラブルシューティングとエラー処理 ー 避けるべきコーディング方法の一覧を示し、UNIX® シグナル処理のエラーを扱うための特別な関数について説明します。

注釈 **共有ライブラリとコールアウト・ライブラリ**

このドキュメントでは、共有ライブラリという用語は、動的にリンクされたファイル (Windows の DLL ファイル、または UNIX® および関連オペレーティング・システムの SO ファイル) を示します。コールアウト・ライブラリは、$ZF コールアウト・インタフェースへのフックを含む共有ライブラリであり、多様な $ZF 関数が実行時にこれをロードし、アクセスできるようにします。

4.1 コールアウト・ライブラリの概要

ObjectScript コードからコールアウト・ライブラリにアクセスするには複数の方法がありますが、一般的には、ライブラリ名、関数名、および必要な引数を指定するのが主な方法です (“コールアウト・ライブラリ関数の呼び出し” を参照してください)。例えば、以下のコードでは、簡単なコールアウト・ライブラリ関数を呼び出します。

コールアウト・ライブラリ simplecallout.dll からの関数 AddInt の呼び出し

以下の ObjectScript コードは、ターミナルで実行されます。これは、simplecallout.dll という名前のコールアウト・ライブラリをロードし、2 つの整数引数を加算して合計を返す AddInt と言う名前のライブラリ関数を呼び出します。

```
USER> set sum = $ZF(-3,"simplecallout.dll","AddInt",2,2)
USER> write "The sum is ",sum,!
The sum is 4
```

この例では、$ZF(-3) を使用します。これは、単一のコールアウト・ライブラリ関数を呼び出す最も簡単な方法です。その他のオプションは、“コールアウト・ライブラリ関数の呼び出し”を参照してください。

simplecallout.dll コールアウト・ライブラリの複雑さは、それを呼び出すコードとあまり変わりません。すべてのコールアウト・ライブラリで必要とされる以下の 3 つの要素を含みます。

1. iris-cdzf.h コールアウト・ヘッダ・ファイルを含めるときに提供される標準コード。
2. パラメータが正しく指定された 1 つ以上の関数。
3. ZFEntry テーブル のマクロ・コード。これは、ライブラリがロードされる際に InterSystems IRIS がコールアウト関数を見つけるために使用するメカニズムを生成します (詳細は、“ZFEntry テーブルの作成”を参照してください)。

simplecallout.dll コールアウト・ライブラリを生成するためにコンパイルされたコードは以下のとおりです。

simplecallout.dll のコールアウト・コード

```
#define ZF_DLL  /* Required for all Callout code. */
#include <iris-cdzf.h>  /* Required for all Callout code. */

int AddTwoIntegers(int a, int b, int *outsum) {
   *outsum = a+b;   /* set value to be returned by the $ZF function call */
   return IRIS_SUCCESS;   /* set the exit status code */
}
ZFBEGIN
   ZFENTRY("AddInt","iiP",AddTwoIntegers)
ZFEND
```

- 最初の 2 行では、ZF_DLL を定義し、iris-cdzf.h ファイルを含める必要があります。これらの 2 行は必ず含める必要があります。
- AddTwoIntegers() 関数が次に定義されます。これには、以下の機能があります。
  - 2 つの入力パラメータである整数 a と b、および 1 つの出力パラメータである整数ポインタ *outsum。
  - 出力パラメータ *outsum に値を割り当てる文。これは、$ZF(-3) への呼び出しによって返される値です。
  - `return` 文は、関数出力値を返しません。代わりに、$ZF 呼び出しが正常に行われた場合に InterSystems IRIS によって受け取られる終了ステータス・コードを指定します。この関数が失敗した場合、InterSystems IRIS はシステムによって生成された終了ステータス・コードを受け取ります。
- 最後の 3 行は、InterSystems IRIS がコールアウト・ライブラリ関数を見つけるために使用する ZFEntry テーブルを生成するマクロ呼び出しです。この例では、単一のエントリのみです。ここで、
  - `"AddInt"` は、$ZF 呼び出し内の関数を識別するために使用される文字列です。
  - `"iiP"` は、2 つの入力値および出力値のデータ型を指定する文字列です。
  - `AddTwoIntegers` は、C 関数のエントリ・ポイント名です。

ZFEntry テーブルは、共有ライブラリをロードし、$ZF コールアウト・インタフェースによりアクセス可能にするメカニズムです (“ZFEntry テーブルの作成”を参照)。ZFENTRY 宣言は、C 関数と ObjectScript $ZF 呼び出し間のインタフェースを指定します。この例では、このインタフェースは、次のように動作します。

- C 関数宣言 は、以下の 3 つのパラメータを指定します。

```
int AddTwoIntegers(int a, int b, int *outsum)
```

パラメータ a および b は入力であり、outsum は、出力値を受け取ります。AddTwoIntegers の返り値は、終了ステータス・コードであり、出力値ではありません。

- *ZFENTRY* マクロ は、InterSystems IRIS 内での関数の識別方法、およびパラメータが渡される方法を定義します。

  ```
  ZFENTRY("AddInt","iiP",AddTwoIntegers)
  ```

  "AddInt" は、$ZF 呼び出し内の C 関数 AddTwoIntegers を指定するために使用されるライブラリ関数識別子です。リンク宣言 ("iiP") は、パラメータ a および b をリンク・タイプ i (入力のみの整数) として、outsum をリンク・タイプ P (入力および出力の両方に使用できる整数ポインタ) として宣言します。

- $ZF(-3) 関数呼び出し は、ライブラリ名、ライブラリ関数識別子、および入力パラメータを指定し、出力パラメータの値を返します。

  ```
  set sum = $ZF(-3,"simplecallout.dll","AddInt",2,2)
  ```

  パラメータ a および b は、最後の 2 つの引数によって指定されます。3 番目のパラメータ outsum は出力にのみ使用されるため、引数は必要ありません。outsum の値は、$ZF(-3) 呼び出しが返されたときに sum に割り当てられます。

4.1.1 ZFEntry テーブルの作成

すべてのコールアウト・ライブラリは、ZFEntry テーブルを定義する必要があり、これにより InterSystems IRIS がコールアウト関数をロードしてアクセスできるようになります (簡単な例は “コールアウト・ライブラリの概要” を参照してください)。ZFEntry テーブルは ZFBEGIN で始まり ZFEND で終わるマクロ・コードのブロックによって生成されます。これら 2 つのマクロ間で、関数を公開するごとに 1 回、ZFENTRY マクロを呼び出す必要があります。

各 ZFENTRY 呼び出しは、以下の 3 つの引数をとります。

```
ZFENTRY(zfname,linkage,entrypoint)
```

ここで、zfname は $ZF 呼び出し内で関数を指定するために使用される文字列、linkage は引数を渡す方法を指定する文字列、entrypoint は C 関数のエントリ・ポイント名です。

コールアウト・ライブラリを作成するには、ライブラリ関数を見つけるための内部の GetZFTable 関数を生成するスイッチである #define ZF_DLL 指示文をコードに含める必要があります。コールアウト・ライブラリがロードされると、InterSystems IRIS はこの関数を呼び出して後続のライブラリ関数名の検索用にライブラリを初期化します。

注釈 **ZFEntry のシーケンス番号**

ZFEntry テーブル内のエントリの位置が重要となる場合があります。$ZF(-5) インタフェースおよび $ZF(-6) インタフェース (“コールアウト・ライブラリ関数の呼び出し” で説明します) は、両方ともテーブル内でシーケンス番号 (1 で始まる) を指定することによってライブラリ関数を呼び出します。例えば、$ZF(-6) は以下の呼び出しを使用して ZFEntry テーブル内の 3 番目の関数を呼び出します。

```
x = $ZF(-6,libID,3)
```

ここで libID はライブラリ識別子、3 はテーブル内の 3 番目のエントリのシーケンス番号です。

注釈 **プリコンパイル済みのヘッダ**

一部のコンパイラ (Microsoft Visual Studio など) は、プリコンパイル済みのヘッダをサポートします。プリコンパイル済みのヘッダを使用する場合、#define ZF_DLL 文がプリコンパイルに対して有効である必要があります。そうでない場合、結果として生成される dll により、使用時に <DYNAMIC LIBRARY LOAD> エラーが発生します。コールアウト・ライブラリにはプリコンパイル済みのヘッダを使用しないことを強くお勧めします。

4.2 ZFEntry リンク・オプション

各 ZFENTRY 文 (“ZFEntry テーブルの作成” を参照) では、関数引数が渡される方法を決定する文字列が必要です。このセクションでは、利用可能なリンク・オプションについて詳細に説明します。

- リンクの概要 – さまざまなリンク・タイプの概要を示し、この章で説明するすべてのリンク・オプションをリストします。
- 数値リンクの使用 – 数値パラメータのリンク・オプションを説明します。
- NULL で終了する文字列を C リンク・タイプで渡す – NULL で終了する文字列のリンク・オプションについて説明します。
- 短い計算文字列を B リンク・タイプで渡す – 計算文字配列に ZARRAY 構造を使用するリンクについて説明します。
- 標準的な計算文字列を J リンク・タイプで渡す – 計算文字配列に InterSystems IRIS IRIS_EXSTR 構造を使用するリンクについて説明します。
- 従来の短い文字列の $ZF ヒープの構成 – 従来の短い文字列パラメータ渡しのメモリ割り当てを制御する InterSystems IRIS システム設定について説明します。

4.2.1 リンクの概要

各 ZFENTRY 文 (“ZFEntry テーブルの作成” を参照) では、引数が渡される方法を説明する文字列が必要です。例えば、"iP" は、整数と整数へのポインタの 2 つのパラメータを指定します。2 番目の文字は、2 番目の引数が入力と出力の両方に使用されることを指定するために大文字になっています。コードは、最大で 32 の実パラメータおよび仮パラメータを持つことができます。

大文字のリンク・タイプを指定する場合 (i を除くすべてのリンク・タイプで許可されている)、入力と出力の両方にこの引数を使用することができます。1 つの出力引数のみが指定されている場合は、その最終値は関数の返り値として使用されます。複数の出力引数が指定されている場合は、すべての出力引数がコンマ区切りの文字列として返されます。

出力引数は入力引数として使用する必要はありません。すべての入力引数の後に出力限定引数を指定する場合、関数は出力引数のいずれも指定することなく呼び出すことができます (例は、“コールアウト・ライブラリの概要” を参照してください)。

ObjectScript プログラマの観点からすると、パラメータは入力のみです。実パラメータの値は $ZF 呼び出しによって評価され、C ルーチン宣伝内の仮パラメータにリンクされます。C 仮パラメータへの変更は失われるか、または$ZF 返り値にコピー可能になるかのいずれかです。

ZFENTRY マクロが、仮パラメータを返り値として使用するように指定しない場合は、$ZF 呼び出しは空の文字列 ("") を返します。リンク宣言は、1 つ以上の出力パラメータを含むことができます。この場合、すべての返り値は単一のコンマ区切り文字列に変換されます。複数の返り値間に挿入されたコンマと、1 つの返り値内に存在するコンマとを区別する方法はないため、最後の返り値のみがコンマを含むようにします。

以下のテーブルは、利用可能なオプションを説明しています。

C データ型	Input	In/Out	メモ
int	i または 4i	なし (P を使用)	32 ビットの整数を指定します。i リンク・タイプは入力のみです。整数タイプを返すには、P または 4P (int *) を使用します。入力引数は数値文字列の場合があります (メモ 1 参照)。
int *	p または 4p	P または 4P	32 ビットの整数へのポインタです。入力引数は数値文字列の場合があります (メモ 1 参照)。

C データ型	Input	In/Out	メモ
_int64	8i	なし (8P を使用)	64 ビットの整数を指定します。64 ビットの整数タイプを返すには、8P を使用します。入力引数は数値文字列の場合があります (メモ 1 参照)。
_int64 *	8p	8P	64 ビットの整数へのポインタです。入力引数は数値文字列の場合があります (メモ 1 参照)。
double *	d	D	入力引数は数値文字列の場合があります (メモ 1 参照)。#D を使用して double * を基数 2 形式で保持します (メモ 2 を参照)。
float *	f	F	入力引数は数値文字列の場合があります (メモ 1 参照)。#F を使用して float * を基数 2 形式で保持します (メモ 2 を参照)。
char *	1c または c	1C または C	これは、一般的な C NULL 終了文字列です (メモ 3 を参照)。
unsigned short *	2c または w	2C または W	これは、C 形式の NULL 終了 UTF-16 文字列です (メモ 3 を参照)。
wchar t *	4c	4C	これは、wchar_t 要素のベクトルとして保存される C 形式の NULL 終了文字列です (メモ 3 および 4 を参照)。
ZARRAYP	1b または b	1B または B	最大 32,767 文字の短い 8 ビット各国語文字列。
ZWARRAYP	2b または s	2B または S	最大 32,767 文字の短い 16 ビット Unicode 文字列。
ZHARRAYP	4b	4B	wchar_t によって実装される要素内に保存される、最大 32,767 文字の短い Unicode 文字列 (メモ 4 を参照してください)
IRIS_EXSTR	1j または j	1J または J	8 ビット各国語文字で構成される、標準的な文字列 (文字列長の制限あり)
IRIS_EXSTR	2j または n	2J または N	16 ビット Unicode 文字で構成される、標準的な文字列 (文字列長の制限あり)
IRIS_EXSTR	4j	4J	wchar_t 文字で構成される、標準的な文字列 (文字列長の制限あり) (メモ 4 を参照)

1. i、p、d、f ー 数値引数が指定されると、InterSystems IRIS では文字列の入力引数が許可されます。詳細は、“数値リンクの使用”を参照してください。
2. #F、#D ー 数値を基数 2 浮動小数点形式で保持するには、float * で #F を使用するかまたは double * で #D を使用します。詳細は、“数値リンクの使用”を参照してください。
3. 1C、2C、4C ー このリンクで渡されるすべての文字列は最初の NULL 文字で切り捨てられます。詳細は、“NULL で終了する文字列を C リンク・タイプで渡す”を参照してください。
4. 4B、4C、4J ー wchar_t は通常 32 ビットですが、InterSystems IRIS では各 Unicode 文字を格納するのに 16 ビットのみを使用します。大きい wchar_t 値を含む出力引数は、$ZF の戻り値に割り当てるために UTF-16 に変換されます。UTF-16 を含む文字列 (サロゲート・ペア) は、$ZF 入力引数のために wchar_t に展開されます。wchar_t の真の値には、ObjectScript 関数 $WASCII() および $WCHAR() を使用してアクセスできます。

構造体と引数のプロトタイプ定義 (インターシステムズ内部定義を含む) は、インクルード・ファイル iris-cdzf.h にあります。

4.2.2 数値リンクの使用

数値リンク・タイプは、以下のデータ型用に提供されています。

C データ型	Input	In/Out	メモ
int	i	なし (P を使用)	32 ビットの整数を指定します。i リンク・タイプは入力のみです。整数タイプを返すには、P または 4P (int *) を使用します。
int *	p	P	32 ビットの整数へのポインタです。
_int64	8i	なし (8P を使用)	64 ビットの整数を指定します。64 ビットの整数タイプを返すには、8P を使用します。
_int64 *	8p	8P	64 ビットの整数へのポインタです。
double *	d	D	#D (出力のみ) を使用して double * を基数 2 形式で返します。
float *	f	F	#F (出力のみ) を使用して float * を基数 2 形式で返します。

数値引数が指定されると、InterSystems IRIS では文字列の入力引数が許可されます。文字列が渡されると、先頭の数字が文字列から解析されて数値が派生します。先頭の数字がない場合は、値 0 が受け取られます。つまり、"2DOGS" は 2.0 として受け取られますが、"DOG" は 0.0 として受け取られます。整数の引数は切り捨てられます。例えば、"2.1DOGS" は 2 として受け取られます。詳細は、"ObjectScript の使用法" の “文字列から数値への変換” を参照してください。

注釈 **浮動小数点数での精度の保持**

出力リンクが F (float *) または D (double *) で指定されている場合、返す数値は内部基数 10 の数の形式に変換されます。数値を基数 2 形式で保持するには、float * で #F を使用するかまたは double * で #D を使用します。

入力引数では # 接頭語は許可されていません。変換 (精度が若干失われる原因となる場合があります) を回避するには、関数を呼び出す ObjectScript コード内で $DOUBLE を使用して入力値を作成する必要があり、対応する入力リンクは小文字の f または d として指定する必要があります。

InterSystems IRIS は、標準の IEEE 形式の 64 ビット浮動小数点を作成する $DOUBLE 関数をサポートします。これらの数は、精度を失うことなく外部関数と InterSystems IRIS 間で渡すことができます (外部関数が 64 ビットの double ではなく 32 ビットの float を使用する場合を除く)。出力では、IEEE 形式の使用は、接頭語 # を F または D の引数タイプへ追加すると、指定できます。例えば、"i#D" は整数の入力引数 1 つと、64 ビット浮動小数点出力引数 1 つに引数リストを指定します。

4.2.3 NULL で終了する文字列を C リンク・タイプで渡す

このリンク・タイプは、InterSystems IRIS が NULL ($CHAR(0)) 文字を含む文字列を送信しないことがわかっている場合にのみ使用します。このデータ型を使用する際は、C 関数は、文字列が長い場合でも、InterSystems IRIS によって渡される文字列を最初の NULL 文字で切り捨てます。例えば、文字列 "ABC"_$CHAR(0)_"DEF" は、"ABC" に切り捨てられます。

C データ型	Input	In/Out	メモ
char *	1c または c	1C または C	これは、一般的な C NULL 終了文字列です。
unsigned short *	2c または w	2C または W	これは、C 形式の NULL 終了 UTF-16 文字列です。
wchar t *	4c	4C	これは、wchar_t 要素のベクトルとして保存される C 形式の NULL 終了文字列です。

数値文字列を返すために 3 つすべてのリンク・タイプを使用する短いコールアウト・ライブラリを以下に示します。

C リンクを使用した NULL 終了文字列渡し

以下の 3 つの関数それぞれは、ランダム整数を生成して最大 6 桁の数値文字列に変換し、C リンクを使用してその文字列を返します。

```
#define ZF_DLL   // Required when creating a Callout library.
#include <iris-cdzf.h>
#include <stdio.h>
#include <wchar.h>   // Required for 16-bit and 32-bit strings

int get_sample(char* retval) {  // 8-bit, null-terminated
   sprintf(retval,"%d",(rand()%1000000));
   return ZF_SUCCESS;
}

int get_sample_W(unsigned short* retval) {  // 16-bit, null-terminated
   swprintf(retval,6,L"%d",(rand()%1000000));
   return ZF_SUCCESS;
}

int get_sample_H(wchar_t* retval) {  // 32-bit, null-terminated
   swprintf(retval,6,L"%d",(rand()%1000000));
   return ZF_SUCCESS;
}

ZFBEGIN
ZFENTRY("GetSample","1C",get_sample)
ZFENTRY("GetSampleW","2C",get_sample_W)
ZFENTRY("GetSampleH","4C",get_sample_H)
ZFEND
```

4.2.4 短い計算文字列を B リンク・タイプで渡す

iris-cdzf.h コールアウト・ヘッダ・ファイルは、短い文字列（インターシステムズの従来の文字列型）を表す計算文字列構造 ZARRAY、ZWARRAY、および ZHARRAY を定義します。これらの構造は、文字要素の配列（それぞれ、8 ビット、16 ビット Unicode、または 32 ビット wchar t）および配列内の要素の数を指定する short 整数（最大値は 32,768）を含みます。以下はその例です。

```
typedef struct zarray {
    unsigned short len;
    unsigned char data[1]; /* 1 is a dummy value */
   } *ZARRAYP;
```

以下はその説明です。

- len ー 配列の長さが格納されます。
- data ー 文字データを格納する配列です。要素タイプは、ZARRAY は unsigned char、ZWARRAY は unsigned short、ZHARRAY は wchar_t です。

B リンクは、3 つの配列構造に対応して、ポインタ型 ZARRAYP、ZWARRAYP、および ZHARRAYP を指定します。返される配列の最大サイズは 32,767 文字です。

C データ型	Input	In/Out	メモ
ZARRAYP	1b または b	1B または B	8 ビットの最大 32,767 文字を含む短い各国語文字列。
ZWARRAYP	2b または s	2B または S	16 ビットの最大 32,767 文字を含む短い Unicode 文字列。
ZHARRAYP	4b	4B	最大 32,767 の wchar_t 文字を含む短い Unicode 文字列。

引数の最大合計長さは、1 文字あたりのバイト数に依存します (“$ZF ヒープの構成” を参照)。

数値文字列を返すために 3 つすべてのリンク・タイプを使用するコールアウト・ライブラリを以下に示します。

B リンクを使用した計算文字列渡し

以下の 3 つの関数それぞれは、ランダム整数を生成して最大 6 桁の数値文字列に変換し、B リンクを使用して文字列を返します。

```
#define ZF_DLL   // Required when creating a Callout library.
#include <iris-cdzf.h>
#include <stdio.h>
#include <wchar.h>   // Required for 16-bit and 16-bit characters

int get_sample_Z(ZARRAYP retval) {  // 8-bit, counted
   unsigned char numstr[6];
   sprintf(numstr,"%d",(rand()%1000000));
   retval->len = strlen(numstr);
   memcpy(retval->data,numstr,retval->len);
   return ZF_SUCCESS;
}

int get_sample_ZW(ZWARRAYP retval) {  // 16-bit, counted
   unsigned short numstr[6];
   swprintf(numstr,6,L"%d",(rand()%1000000));
   retval->len = wcslen(numstr);
   memcpy(retval->data,numstr,(retval->len*sizeof(unsigned short)));
   return ZF_SUCCESS;
}

int get_sample_ZH(ZHARRAYP retval) {  // 32-bit, counted
   wchar_t numstr[6];
   swprintf(numstr,6,L"%d",(rand()%1000000));
   retval->len = wcslen(numstr);
   memcpy(retval->data,numstr,(retval->len*sizeof(wchar_t)));
   return ZF_SUCCESS;
}


ZFBEGIN
ZFENTRY("GetSampleZ","1B",get_sample_Z)
ZFENTRY("GetSampleZW","2B",get_sample_ZW)
ZFENTRY("GetSampleZH","4B",get_sample_ZH)
ZFEND
```

注釈　複数の値を含む出力引数文字列ではコンマが区切り文字として使用されます。コンマは計算文字列配列の一部となるため、引数リストの 最後 にこれらの配列を宣言し、呼び出しごとに 1 つの配列を使用します。

4.2.5 標準的な計算文字列を J リンク・タイプで渡す

iris-callin.h ヘッダ・ファイルは、標準的な InterSystems IRIS 文字列を表す計算文字列構造 IRIS_EXSTR を定義します。この構造は、文字要素の配列 (8 ビット、16 ビット Unicode、または 32 ビット wchar_t) および配列内の要素の数を指定する int 値 (文字列長の制限あり) を含みます。

```
typedef struct {
   unsigned int   len;         /* length of string */
   union {
      Callin_char_t  *ch;      /* text of the 8-bit string */
      unsigned short *wch;     /* text of the 16-bit string */
      wchar_t        *lch;     /* text of the 32-bit string */
/* OR unsigned short *lch   if 32-bit characters are not enabled */
   } str;
} IRIS_EXSTR, *IRIS_EXSTRP;
```

C データ型	Input	In/Out	メモ
IRIS_EXSTR	1j または j	1J または J	8 ビット各国語文字の標準的な文字列
IRIS_EXSTR	2j または n	2J または N	16 ビット Unicode 文字の標準的な文字列
IRIS_EXSTR	4j	4J	32 ビット wchar_t 文字の標準的な文字列

IRIS_EXSTR データ構造は、コールイン API から関数で操作します (低レベルの InterSystems 関数呼び出しのライブラリ。詳細は、"コールイン API の使用法" の "コールイン関数リファレンス" を参照してください。名前はよく似ていますが、コールイン API と $ZF コールアウト・インタフェースはまったく別の製品です)。

IRIS_EXSTR のインスタンスを作成および破棄するには、以下の関数を使用します。

- IrisExStrNew[W][H] ー 文字列に対して要求されたストレージの容量を割り当て、IRIS_EXSTR 構造に、構造の長さおよび構造の値フィールドへのポインタを埋め込みます。
- IrisExStrKill ー IRIS_EXSTR 文字列に関連付けられているストレージを解放します。

数値文字列を返すために 3 つすべてのリンク・タイプを使用するコールアウト・ライブラリを以下に示します。

J リンクを使用した文字列渡し

以下の 3 つの関数それぞれは、ランダム整数を生成して最大 6 桁の数値文字列に変換し、J リンクを使用して文字列を返します。

```
#define ZF_DLL   // Required when creating a Callout library.
#include <iris-cdzf.h>
#include <stdio.h>
#include <wchar.h>
#include <iris-callin.h>

int get_sample_L(IRIS_EXSTRP retval) {  // 8-bit characters
   Callin_char_t numstr[6];
   size_t len = 0;
   sprintf(numstr,"%d",(rand()%1000000));
   len = strlen(numstr);
   IRISEXSTRKILL(retval);
   if (!IRISEXSTRNEW(retval,len)) {return ZF_FAILURE;}
   memcpy(retval->str.ch,numstr,len);   // copy to retval->str.ch
   return ZF_SUCCESS;
}

int get_sample_LW(IRIS_EXSTRP retval) {  // 16-bit characters
   unsigned short numstr[6];
   size_t len = 0;
   swprintf(numstr,6,L"%d",(rand()%1000000));
   len = wcslen(numstr);
```

```
   IRISEXSTRKILL(retval);
   if (!IRISEXSTRNEW(retval,len)) {return ZF_FAILURE;}
   memcpy(retval->str.wch,numstr,(len*sizeof(unsigned short)));   // copy to retval->str.wch
   return ZF_SUCCESS;
}

int get_sample_LH(IRIS_EXSTRP retval) {  // 32-bit characters
   wchar_t numstr[6];
   size_t len = 0;
   swprintf(numstr,6,L"%d",(rand()%1000000));
   len = wcslen(numstr);
   IRISEXSTRKILL(retval);
   if (!IRISEXSTRNEW(retval,len)) {return ZF_FAILURE;}
   memcpy(retval->str.lch,numstr,(len*sizeof(wchar_t)));   // copy to retval->str.lch
   return ZF_SUCCESS;
}

ZFBEGIN
ZFENTRY("GetSampleL","1J",get_sample_L)
ZFENTRY("GetSampleLW","2J",get_sample_LW)
ZFENTRY("GetSampleLH","4J",get_sample_LH)
ZFEND
```

注釈 **IRIS_EXSTRP 入力引数の常時削除**

前述の例では、**IRISEXSTRKILL(retval)** は、メモリから入力引数を削除するために常に呼び出されています。これは、引数が出力に使用されない場合でも常に実行する必要があります。実行しない場合は、メモリ・リークが発生します。

4.2.6 従来の短い文字列の $ZF ヒープの構成

注釈 このセクションは、従来の短い文字列（“短い計算文字列を B リンク・タイプで渡す”を参照してください）にのみ適用されます。標準的な InterSystems IRIS 文字列（“標準的な計算文字列を J リンク・タイプで渡す”を参照してください）は、独自のスタックを使用します。

$ZF ヒープは、すべての $ZF の短い文字列入力および出力パラメータに割り当てられる仮想メモリ領域です。これは、以下の InterSystems IRIS システム設定によって制御されます。

- ZFString は、1 つの文字列パラメータに対して使用が許可される文字数です。このために実際に必要なバイト数は、使用している文字が 8 ビット文字か、16 ビット Unicode 文字か、UNIX® 上の 32 ビット文字かによって異なります。この設定で許可されている範囲は、**0** から **32767** 文字です。既定は、**0** であり、最大値を使用することを示します。
- ZFSize は、InterSystems IRIS がすべての $ZF 入力および出力パラメータに割り当てる合計バイト数です。この設定で許可されている範囲は、**0** から **270336** バイトです。**0**（既定の設定）は、InterSystems IRIS が ZFString の値に基づいて適切な値を計算することを示します。

以下のように ZFSize（合計バイト数）を ZFString（文字列あたりの最大文字数）に基づいて計算します。

```
ZFSize = (<bytes per character> * ZFString) + 2050
```

例えば、ZFString の値が既定の 32767 文字だとします。

- Unicode 16 ビット文字を使用すると、ZFSize の適切な値は **(2 * 32767 + 2050) = 67584** バイトになります。
- UNIX® 32 ビット文字を使用すると、ZFSize の適切な値は **(4 * 32767 + 2050) = 133118** バイトになります。

これらの設定は、以下のいずれかの場所で変更できます。

- 構成パラメータ・ファイル（″構成パラメータ・ファイル・リファレンス″の“[config]”セクションの“zfheap”を参照してください）。
- 管理ポータル（″追加構成設定リファレンス″の“詳細メモリ設定”にある ZFSize および ZFString のエントリを参照してください）。

4.3 互換性のある言語とコンパイラ

$ZF コールアウト・インタフェースを使用して、外部言語で関数を記述し、それらを ObjectScript から呼び出すことができます。コールアウト・ライブラリは、通常は C で記述されますが、ご使用の C コンパイラで認識される呼び出し規則を使用するその他のコンパイル言語で記述される場合もあります。互換性に関する 2 つの問題が発生します。1 つ目は、コンパイラは C と互換性があるアプリケーション・バイナリ・インタフェース (ABI) を使用する必要があることです。2 つ目は、コンパイラは InterSystems IRIS と互換性がないランタイム・ライブラリ機能に依存しないコードを生成する必要があることです。

インターシステムズは、すべてのプラットフォーム上で InterSystems IRIS を生成するために使用するのと同じ C コンパイラの使用をサポートします。

プラットフォーム	コンパイラ
IBM AIX	AIX 用 IBM XL C
Mac OS X (Darwin)	Xcode
Microsoft Windows	Microsoft Visual Studio
Linux (系列すべて)	GNU プロジェクト GCC C

ほとんどのプラットフォームは、標準化されたアプリケーション・バイナリ・インタフェース (ABI) を備え、ほとんどのコンパイラに互換性があります。Intel x86-32 および x86-64 プラットフォームは主な例外であり、これらのプラットフォームには、複数の呼び出し規則が存在します。これらのプラットフォームでの呼び出し規則については、(https://en.wikipedia.org/wiki/X86_calling_conventions) を参照してください。

多くの C コンパイラでは、外部ルーチンに対して異なる呼び出し規則を宣言することを許可しています。適切な呼び出し規則を宣言する C ラッパ・ルーチンを記述することによって、別の言語で記述されたルーチンを呼び出すことができます。

4.4 コールアウト・ライブラリの Runup 関数と Rundown 関数

InterSystems コールアウト・ライブラリは、共有オブジェクトがロードされたとき (runup) またはアンロードされたとき (rundown) に呼び出されるカスタム内部関数を含むことができます。どちらのケースも引数は渡されません。関数は以下のように使用されます。

- ZFInit ー コールアウト・ライブラリが $ZF(-3)、$ZF(-4,1)、または $ZF(-6) によって最初にロードされたときに呼び出されます。この関数からの返りコードは、エラーがないことを示すゼロ、もしくはいくつかの障害の存在を示すゼロ以外の値になります。呼び出しが正常に実行されると、ZFUnload rundown 関数のアドレスが保存されます。
- ZFUnload ー コールアウト・ライブラリが $ZF(-3) への呼び出しによってアンロードまたは置換されたとき、あるいは $ZF(-4,2) または $ZF(-4,4) によってアンロードされたときに呼び出されます。プロセスの停止では呼び出されません。エラーが rundown 関数実行中に発生した場合、それ以降、コールアウト・ライブラリのアンロードを許可するための呼び出しは無効となります。ZFUnload からの返り値は現在無視されています。

コールアウト・ライブラリを構築する際、リンクを確立するプロシージャの間に、ZFInit と ZFUnload 記号を明示的にエクスポートする必要がある場合があります。

4.5 トラブルシューティングとエラー処理

このセクションでは、以下の項目について説明します。

- ワースト・プラクティス ― 深刻な問題を引き起こす可能性のあるプラクティスをいくつか示します。
- UNIX® シグナル処理のエラーの処理 ― プロセスがシグナルを受信したときに生じる可能性のある失敗したシステム呼び出しから回復するのに役立ついくつかの関数について説明します。

4.5.1 ワースト・プラクティス

ほぼすべてのルーチンは $ZF コールアウト・インタフェースで呼び出すことができますが、$ZF コールアウト・インタフェースは数学関数で使用するのが最適です。また、InterSystems IRIS I/O でうまく処理されない外部デバイスへのインタフェース、または InterSystems IRIS インタフェースが存在しないシステム・サービスの場合にも効果的に使用することができます。

以下の動作によって重大な問題が発生する可能性があります。

- 自身に属していないメモリへのアクセス

  メモリ・アクセス違反は、InterSystems IRIS によって処理され、InterSystems IRIS 内のバグとして扱われます。

- トラップによって処理されるその他のエラーの発生

  トラップによって処理されるエラー (ほとんどのプラットフォーム上でのゼロによる除算など) も InterSystems IRIS 内のバグとして扱われます。

- プロセスの優先度の変更

  InterSystems IRIS は、InterSystems IRIS を実行する他のプロセスと対話する必要があります。優先度を下げると、上げるのと同様に問題が発生する場合があります。例えば、プロセスが CPU を解放する直前にスピン・ロックで保護されたリソースを取得することを考えてみましょう。優先度が低すぎると、高い優先度のその他のプロセスがリソースを奪い合い、事実上、スピン・ロックを解除できるようにするプロセスの実行を妨げます。

- 中断のマスク

  インタロックを実装するために中断を短時間マスクする場合がありますが、中断をマスクしたままにしないように十分注意する必要があります。

- 削除できないリソースを作成または開く

  ファイルを開いて malloc を使用してメモリを割り当てることは問題ありません。それらのリソースはプロセスの終了時に閉じられるかまたは解放されます。2 番目のスレッドを作成すると、2 番目のスレッドが InterSystems IRIS プロセスの終了前に正常に終了することを保証できないため、2 番目のスレッドは作成しないでください。

- opaque 以外のオブジェクトを ObjectScript 以外のコードで返す

  コード内のメモリのブロックを malloc して、それを読み取るために `$VIEW(address,-3,size)` を使用できると考えないでください。また、ObjectScript 以外のコードに malloc ブロックを戻さないでください。コードは opaque ハンドルを返し、後で opaque ハンドルを受け取った時に使用する前に有効であることを確認する必要があります。

- プロセスの終了

  `exit` だけを呼び出すことはしないでください。常に `ZF_SUCCESS` または `ZF_FAILURE` のいずれかと共に返します (これらの値の実装は InterSystems IRIS プラットフォーム間で異なることに留意してください)。

- exec のいずれかのバージョンの呼び出しによる終了

  子プロセス内で fork してから exec を呼び出すことはできますが、必ず親は InterSystems IRIS に返り、子プロセスは InterSystems IRIS に返らないようにしてください。

- プロセスでのエラー処理の振る舞いの変更

  Windows とは異なり、UNIX® システムでは、現在の内部フレームのローカル・エラー処理のみを確立できます。

4.5.2 UNIX® シグナル処理のエラーの処理

UNIX® および関連するオペレーティング・システムで実行中の場合、システム呼び出しの中には、open、read、write、close、ioctl、および pause などのシグナルをプロセスが受け取ると、失敗するものがあります。関数がこれらのシステム呼び出しを使用する場合、実際のエラー、Ctrl-C、再起動を必要とする呼び出しと区別できるように、コードを記述する必要があります。

以下の関数により、非同期のイベントの確認や $ZF でのアラーム・ハンドラの設定ができます。関数の宣言は、iris-cdzf.h に組み込まれます。

sigrtclr()

`int sigrtclr();` ー 再試行フラグを消去します。sigrtchk() を使用する前に一度呼び出す必要があります。

dzfalarm()

`int dzfalarm();` ー 新規 SIGALRM ハンドラを構築します。

$ZF にエントリする際に、前のハンドラが自動的に保存されます。終了すると、自動的にリストアされます。ユーザ・プログラムで、他のシグナル処理を変更しないでください。

sigrtchk()

`int sigrtchk();` ー 非同期イベントをチェックします。これは、open、close、read、write、ioctl、または pause のいずれかのシステム呼び出しが失敗した場合や、プロセスがシグナルを受け取ったときに失敗する呼び出しがある場合に、必ず呼び出す必要があります。ユーザの次の動作を示すコードを返します。

- -1 ー シグナルではありません。入出力エラーを調べてください。errno 変数の内容を参照してください。
- 0 ー 他のシグナルです。割り込みが生じた時点から処理を再開してください。
- 1 ー SIGINT/SIGTERM。SIGTERM "return 0" で $ZF を終了します。これらのシグナルは適切にトラップされます。

一部のデバイスの制御に使用される一般的な $ZF 関数は、以下の擬似コードと同様のロジックを使用します。

```
if ((fd = open(DEV_NAME, DEV_MODE)) < 0) {
    Set some flags
    Call zferror
    return 0;
}
```

プロセスがシグナルを受け取ると、open システム呼び出しは失敗する可能性があります。通常、これはエラーではないため、処理を再開できます。しかし、シグナルによっては他の処理が必要な場合もあります。すべての可能性を考慮するために、以下の C コードを使用することを検討してください。

```
sigrtclr();
while (TRUE) {
    if (sigrtchk() == 1) return 1 or 0;
    if ((fd = open(DEV_NAME, DEV_MODE)) < 0) {
        switch (sigrtchk()) {
             case -1:
                /* This is probably a real device error */
                Set some flags
                Call zferror
                return 0;
            case 0:
                /* A innocuous signal was received. Restart. */
                continue;
            case 1:
```

```
                /* Someone is trying to terminate the job. */
                Do cleanup work
                return 1 or 0;
        }
    }
    else break;
}
/*
Code to handle the normal situation:
open() system call succeeded
*/
```

注釈　dzfalarm() を呼び出して新しい SIGALRM ハンドラを確立する場合を除き、エラー処理コードでシグナルの処理を変更しないでください。

5
コールアウト・ライブラリ関数の呼び出し

コールアウト・ライブラリは、$ZF コールアウト・インタフェースへのフックを含む共有ライブラリ (DLL または SO ファイル) であり、多様な $ZF 関数が実行時にロードして、その関数を呼び出すことができるようにします。$ZF コールアウト・インタフェースは、実行時にコールアウト・ライブラリをロードし、そのライブラリから関数を呼び出すために使用できる 4 つの異なるインタフェースを提供します。各インタフェースは、主に、ライブラリの識別方法とメモリへのロード方法が異なります。

- "$ZF() を使用した iriszf コールアウト・ライブラリへのアクセス" では、iriszf という特別な共有ライブラリの使用方法について説明します。このライブラリが利用可能な場合、事前にライブラリをロードしたり、ライブラリ名を指定したりしなくても、`$ZF("funcname",args)` という形式の呼び出しによってその関数にアクセスできます。
- "$ZF(-3) を使用した単純なライブラリ関数呼び出し" では、ライブラリをロードし、ライブラリ・ファイル・パスと関数名を指定することによって関数を呼び出す方法について説明します。使用は簡単ですが、仮想メモリ内で使用できるのは一度に 1 つのライブラリのみです。その他のインタフェースとは異なり、ライブラリ関数を呼び出す前に初期化を行う必要がありません。
- "$ZF(-5) を使用したシステム ID によるライブラリへのアクセス" では、一度に複数のライブラリを効率的に維持し、アクセスするために使用できるインタフェースについて説明します。同時に複数のライブラリをロードでき、それぞれに必要な処理オーバーヘッドは、$ZF(-3) よりはるかに小さくて済みます。メモリ内のライブラリは、ライブラリのロード時に生成されるシステム定義 ID によって識別されます。
- "ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用" では、コールアウト・ライブラリの大規模なセットを処理するための最も効率的なインタフェースについて説明します。このインタフェースは、グローバルに定義されたインデックス・テーブルを介してライブラリへのアクセスを可能にします。インデックスは、InterSystems IRIS のインスタンス内のすべてのプロセスで利用でき、同時に複数のライブラリをメモリに配置できます。インデックス付きの各ライブラリには一意のユーザ定義インデックス番号が指定され、インデックス・テーブルは実行時に定義および変更できます。指定されたライブラリ ID と関連付けられているファイル名は、ライブラリ・ファイルの名前が変更されるとき、または再配置されるときに変更できます。この変更は、インデックス番号でライブラリをロードするアプリケーションに対して透過的です。

5.1 $ZF() を使用した iriszf コールアウト・ライブラリへのアクセス

iriszf という名前のコールアウト・ライブラリがインスタンスの <install_dir>/bin ディレクトリにある場合、関数名と引数のみを指定した $ZF 呼び出しによってその関数を呼び出すことができます (例 : `$ZF("functionName",arg1, arg2)`)。iriszf 関数は、事前にライブラリをロードしなくても呼び出すことができます。また、インスタンス内のすべてのプロセスで利用できます。

カスタムの iriszf ライブラリを定義するには、標準のコールアウト・ライブラリを作成して、それをインスタンスの <install_dir>/bin ディレクトリに移動し、名前を iriszf に変更します (具体的にはプラットフォームに応じて iriszf.dll または iriszf.so)。

以下に、simplecallout.c の例 (“InterSystems コールアウト・ライブラリの作成” を参照) をコンパイルし、それを iriszf ライブラリとして設定する手順を示します。これらの例では、インスタンスが Linux で実行されていて、/intersystems/iris という名前のディレクトリにインストールされていることを想定していますが、手順は基本的にすべてのプラットフォームで同じです。

1. simplecallout.c を作成して保存します。

```
#define ZF_DLL
#include "iris-cdzf.h"
int AddTwoIntegers(int a, int b, int *outsum) {
  *outsum = a+b;  /* set value to be returned by $ZF function call */
  return IRIS_SUCCESS;  /* set the exit status code */
}

ZFBEGIN
  ZFENTRY("AddInt","iiP",AddTwoIntegers)
ZFEND
```

2. コールアウト・ライブラリ・ファイル (simplecallout.so) を生成します。

```
gcc -c -fPIC simplecallout.c -I /intersystems/iris/dev/iris-callin/include/ -o simplecallout.o
gcc simplecallout.o -shared -o simplecallout.so
```

3. InterSystems IRIS ターミナル・セッションから $ZF(-3) を使用してライブラリをテストします。

```
USER>write $ZF(-3,"/mytest/simplecallout.so","AddInt",1,4)
5
```

4. 次に、$ZF() で使用するライブラリをインストールします。simplecallout.so を <install_dir>/bin にコピーし、名前を iriszf.so に変更します。

```
cp simplecallout.so /intersystems/iris/bin/iriszf.so
```

5. InterSystems IRIS セッションから $ZF() を使用してコードを呼び出せることを確認します。

```
USER>write $zf("AddInt",1,4)
5
```

iriszf ライブラリは最初の使用時に 1 回ロードされ、アンロードされることはありません。この処理は、この章で前述した他の $ZF のロードおよびアンロード操作とはまったく無関係です。

注釈 **静的にリンクされたライブラリ**

以前のバージョンの $ZF コールアウト・インタフェースでは、コードをインターシステムズのカーネルに静的にリンクし、$ZF() を使用して呼び出すことができました。静的リンクはサポートされなくなりましたが、iriszf ライブラリで同じ機能が提供されており、カーネルに再リンクする必要はありません。

5.2 単純なライブラリ関数呼び出しでの $ZF(-3) の使用

$ZF(-3) 関数は、コールアウト・ライブラリをロードして、そのライブラリから指定した関数を実行するために使用されます。$ZF(-3) は、1 つのライブラリのみを使用している場合、またはライブラリのロードのオーバーヘッドを懸念して十分な呼

び出しを行っていない場合に、もっとも役立ちます。これにより、ライブラリ名、関数名、および関数引数のコンマ区切りのリストを指定することによって、任意の利用可能なライブラリ関数を呼び出すことができます。

```
result = $ZF(-3, library_name[, function_name[, arguments]])
```

指定されたライブラリは、$ZF(-3) への前の呼び出しによってロードされていない場合にロードされます。一度にロードできるのは 1 つのライブラリのみです。後続の $ZF(-3) 呼び出しが、別のライブラリを指定すると、古いライブラリはアンロードされ、新しいライブラリが置き換わります。このライブラリは、後続の $ZF(-3) 呼び出しが同じライブラリを指定する限りロードされたままになります。ライブラリがロードされた後に、後続の呼び出しで、ライブラリ名を NULL 文字列 (″″) として指定できます。

関数を呼び出すことなくライブラリをロードまたはアンロードすることができます。新しいライブラリをロードするには、ライブラリ名のみを指定します。新しいライブラリをロードせずに現在のライブラリをアンロードするには、NULL 文字列のみを指定します。いずれのケースでも、$ZF(-3) は、ロードまたはアンロードが成功したかどうかを示すステータス・コードを返します。

以下の ObjectScript コードは、2 つの異なるライブラリそれぞれから 2 つの異なる関数を呼び出し、現在のライブラリをアンロードします。

ライブラリのロードおよび関数の呼び出しでの $ZF(-3) の使用

ObjectScript

```
// define Callout library paths
set libOne = "c:\intersystems\iris\bin\myfirstlibrary.dll"
set libTwo = "c:\intersystems\iris\bin\anotherlibrary.dll"

//load and call
SET result1=$ZF(-3,libOne,"FuncA",123)   // loads libOne and calls FuncA
SET result2=$ZF(-3,"","FuncB","xyz")   // calls FuncB from same library

//load, then call with null name
SET status=$ZF(-3,libTwo)   // unloads libOne, loads libTwo
SET result1=$ZF(-3,"","FunctionOne","arg1")
SET result2=$ZF(-3,"","FunctionTwo","argA", "argB")

//unload
SET status=$ZF(-3,"")   // unloads libTwo
```

- 簡便性のため、ライブラリ名は文字列 libOne および libTwo に割り当てられます。
- $ZF(-3) への最初の呼び出しは、コールアウト・ライブラリ libOne をロードし、そのライブラリから関数 FuncA を呼び出します。
- 2 番目の呼び出しは、ライブラリ名に NULL 文字列を指定して、現在ロードされている libOne を再び使用することを示し、そのライブラリから関数 FuncB を呼び出します。
- $ZF(-3) への 3 番目の呼び出しは、ライブラリ名 libTwo のみを指定します。これにより、libOne がアンロードされ、libTwo がロードされますが、ライブラリ関数は呼び出されません。この呼び出しは libTwo が正常にロードされたかどうかを示すステータス・コードを返します。
- 4 番目と 5 番目の呼び出しは、ライブラリ関数 FunctionOne および FunctionTwo を現在ロードされている libTwo から呼び出します。
- 最後の $ZF(-3) 呼び出しは、ライブラリ関数を呼び出さず、ライブラリ名に NULL 文字列を指定します。これにより、libTwo がアンロードされ、新しいライブラリはロードされません。この呼び出しは libTwo が正常にアンロードされたかどうかを示すステータス・コードを返します。

この章の以下のセクションでは、一度に複数のライブラリをロードできる $ZF 関数について説明します。これらの関数は $ZF(-3) と競合しません。$ZF(-3) は、いつでもそれ自体のライブラリのプライベート・コピーをロードおよびアンロードしているかのように使用することができます。

5.3 システム ID によるライブラリへのアクセスでの $ZF(-5) の使用

$ZF(-5) 関数は、ライブラリ関数の呼び出しにシステム定義のライブラリおよび関数の識別子を使用します。多くのライブラリ関数呼び出しを行うアプリケーションでは、これにより処理のオーバーヘッドが大幅に削減できます。複数のライブラリを同時に開くことができます。各ライブラリは、一度のみロードが必要であり、各ライブラリまたは関数識別子は一度のみ生成する必要があります。ユーティリティ関数 $ZF(-4,1)、$ZF(-4,2)、および $ZF(-4,3) は、必要な識別子を取得し、ライブラリをロードまたはアンロードするために使用されます。

- $ZF(-5) ー システム定義のライブラリおよび関数の識別子によって参照される関数を呼び出します。
- $ZF(-4,1) ー ライブラリをロードします。ライブラリ・ファイル名を取得し、ロードされたライブラリのシステム定義ライブラリ ID 値を返します。
- $ZF(-4,2) ー ライブラリ ID によって指定されるコールアウト・ライブラリをアンロードします。
- $ZF(-4,3) ー 指定されたライブラリ ID および関数名の関数 ID 値を返します。

$ZF(-4,1) および $ZF(-4,3) 関数は、コールアウト・ライブラリのロードとライブラリおよび関数識別子の取得に使用されます。$ZF(-4,1) の構文は以下のとおりです。

```
lib_id = $ZF(-4,1,lib_name)   // get library ID
```

ここで lib_name は、共有ライブラリ・ファイルのフルネームとパスであり、lib_id は、返されるライブラリ ID です。$ZF(-4,3) の構文は以下のとおりです。

```
func_id=$ZF(-4,3,lib_id, func_name)   // get function ID
```

ここで lib_id は、ライブラリ ID、func_name は、ライブラリ関数名、func_id は、返される関数 ID 値です。

以下の ObjectScript コードはコールアウト・ライブラリ mylibrary.dll をロードしてライブラリ ID を取得し、"MyFunction" の関数 ID を取得して $ZF(-5) を使用して呼び出します。

$ZF(-5) を使用したライブラリのロードと関数の呼び出し

ObjectScript

```
set libID = $ZF(-4,1,"C:\calloutlibs\mylibrary.dll")
set funcID = $ZF(-4,3,libID, "MyFunction")
set x = $ZF(-5,libID, funcID, "arg1")
```

識別子が定義されると、ライブラリは $ZF(-4,2) によってアンロードされるまでロードされたままになり、その識別子は $ZF(-4,1) または $ZF(-4,3) へのさらなる呼び出しを行うことなく使用することができます。これにより、複数のライブラリからの関数が何度も呼び出される際の処理オーバーヘッドが大幅に削減されます。

以下の ObjectScript コードは 2 つの異なるライブラリをロードし、ロング・ループで両方のライブラリから関数を呼び出します。inputlibrary.dll 内の関数はデータを取得し、outputlibrary.dll 内の関数はデータをプロットおよび保存します。

複数ライブラリおよび多くの関数の呼び出しでの $ZF(-5) の使用

```
Method GraphSomeData(loopsize As %Integer=100000) As %Status
 {
   // load libraries and get system-defined ID values
   set InputLibID = $ZF(-4,1,"c:\intersystems\iris\bin\inputlibrary.dll")
   set OutputLibID = $ZF(-4,1,"c:\intersystems\iris\bin\outputlibrary.dll")
   set fnGetData = $ZF(-4,3,InputLibID,"GetData")
   set fnAnalyzeData = $ZF(-4,3,OutputLibID,"AnalyzeData")
   set fnPlotPoint = $ZF(-4,3,OutputLibID,"PlotPoint")
   set fnWriteData = $ZF(-4,3,OutputLibID,"WriteData")
```

```
   // call functions from each library until we have 100000 good data items
   set count = 0
   do {
      set datapoint = $ZF(-5,InputLibID,fnGetData)
      set normalized = $ZF(-5,OutputLibID,fnAnalyzeData,datapoint)
      if (normalized'="") { set flatdata($INCREMENT(count)) = normalized }
   } while (count<loopsize)
   set status = $ZF(-4,2,InputLibID)   //unload "inputlibrary.dll"

   // plot results of the previous loop and write to output
   for point=1:1:count {
      set list = $ZF(-5,OutputLibID,fnPlotPoint,flatdata(point))
      set x = $PIECE(list,",",1)
      set y = $PIECE(list,",",2)
      set sc = $ZF(-5,OutputLibID,fnWriteData,flatdata(point),x,y,"outputfile.dat")
   }
   set status = $ZF(-4,2,OutputLibID)   //unload "outputlibrary.dll"
   quit 0
}
```

- $ZF(-4,1) への呼び出しでは、コールアウト・ライブラリ inputlibrary.dll および outputlibrary.dll を仮想メモリ内にロードし、それらのシステム定義のライブラリ ID を返します。
- $ZF(-4,3) への呼び出しでは、ライブラリ ID および関数名を使用してライブラリ関数の ID を取得します。返された関数 ID は実際は ZFEntry テーブル・シーケンス番号 (前の章の “ZFEntry テーブルの作成” を参照) です。
- 最初のループでは、$ZF(-5) を使用して各ライブラリから関数を呼び出します。
  - inputlibrary.dll からの GetData() 関数は、未指定のソースから生のデータを読み取ります。
  - outputlibrary.dll からの AnalyzeData() 関数は、生のデータを正規化するかまたは拒否して、空の文字列を返します。
  - 正規化された各 datapoint は、flatdata(count) (ここで ObjectScript 関数 $INCREMENT への最初の呼び出しが count を作成し、1 に初期化する) に保存されます。

  既定では、ループは 100000 の項目をフェッチします。両方のライブラリがロードされメモリ内に残っているため、2 つの異なるライブラリ間を切り替えるための処理オーバーヘッドはありません。
- 最初のループが終了した後に、ライブラリ inputlibrary.dll は必要なくなるため、$ZF(-4,2) が呼び出されてアンロードされます。ライブラリ outputlibrary.dll は、メモリ内に残ります。
- 2 番目のループは、配列 flatdata からの各項目を処理し、未指定の場所にあるファイルに書き込みます。
  - ライブラリ関数 PlotPoint() は、項目を読み取り、プロットされる座標を含むコンマ区切り文字列を返します (ライブラリ関数から複数の出力パラメータが返される方法の説明は、“リンクの概要” を参照してください)。
  - $PIECE 関数は、座標値 x および y を文字列から抽出するために使用されます。
  - ライブラリ関数 WriteData() は、項目および座標を outputfile.dat ファイルに保存します。これは、グラフを出力するために他のアプリケーションで使用されます。
- 2 番目のループが終了した後に、$ZF(-4,2) が再び呼び出されてライブラリ outputlibrary.dll がアンロードされます。

以下のセクションでは、$ZF(-6) インタフェースについて説明します。これは、$ZF(-5) インタフェースと同じ仮想メモリ領域内にライブラリをロードします。

5.4 ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用

$ZF(-6) 関数は、グローバルに定義されたインデックスを介してコールアウト・ライブラリへのアクセスを可能にする、共有ライブラリ・ファイルの場所を認識していないアプリケーションでも使用可能な効率的なインタフェースを提供します。ユーザ定義のインデックス・テーブルは、ライブラリ ID 番号と対応するライブラリ・ファイル名で構成されるキーと値のペアを格納します。指定されたライブラリ ID と関連付けられているファイル名は、ライブラリ・ファイルの名前が変更されたときまたは再配置されたときに変更できます。この変更は、インデックス番号でライブラリをロードするアプリケーションに対して透過的です。その他の $ZF 関数は、インデックス・テーブルを作成して維持するため、および $ZF(-6) によってロードされたライブラリをアンロードするために提供されています。

このセクションでは、以下の $ZF 関数について説明しています。

- $ZF(-6) ー ユーザ指定のインデックス番号によって参照されるコールアウト・ライブラリから関数を呼び出します。ライブラリがまだロードされていない場合には、自動的にロードします。
- $ZF(-4,4) ー インデックス番号によって指定されるコールアウト・ライブラリをアンロードします。
- $ZF(-4,5) および $ZF(-4,6) ー システム・インデックス・テーブル内でエントリを作成または削除します。システム・インデックスは、InterSystems IRIS のインスタンス内のすべてのプロセスでグローバルに利用可能になります。
- $ZF(-4,7) および $ZF(-4,8) ー プロセス・インデックス・テーブル内でエントリを作成または削除します。プロセス・テーブルは、システム・テーブルの前に検索されるため、プロセス内でシステム全体の定義をオーバーライドするために使用します。

$ZF(-6) インタフェースは、$ZF(-5) (“システム ID によるライブラリへのアクセスでの $ZF(-5) の使用” を参照) によって使用されるものと類似していますが、以下の点が異なります。

- $ZF(-6) を使用するには、先にライブラリ・インデックス・テーブルを作成する必要があります。ライブラリ・インデックス値は、ユーザ定義であり、実行時に変更またはオーバーライドできます。
- ライブラリ名は、インデックス内に格納され、ライブラリをロードするアプリケーションによって定義される必要はありません。ライブラリ・ファイルの名前と場所は、インデックス値によって、ライブラリをロードする依存アプリケーションに影響を与えることなくインデックス内で変更することができます。
- ライブラリをロードする別の $ZF 関数はありません。代わりに、ライブラリはその関数の一つを呼び出す最初の $ZF(-6) 呼び出しによって自動的にロードされます。
- 開発者は既にライブラリ関数 ID (ZFEntry テーブル内の順序によって決定される) を知っていると見なされるため、指定された名前およびライブラリ・インデックス値に対して関数 ID を返す $ZF 関数はありません。

以下の例は、$ZF(-6) インタフェースの使用方法を示しています。最初の例は、システム・インデックス・テーブル内にライブラリ ID を定義し、2 番目の例 (別のアプリケーションから呼び出される場合がある) は、ライブラリ ID を使用してライブラリ関数を呼び出します。

$ZF(-4,5) を使用したシステム・インデックス・エントリの定義

この例では、システム・インデックス・テーブル内で `100` を mylibrary.dll のライブラリ ID として設定します。その番号に既に定義が存在している場合は、削除されて置き換えられます。

ObjectScript

```
set LibID = 100
set status=$ZF(-4,4,LibID) // unload any existing library with this ID value
set status = $ZF(-4,5,LibID,"C:\calloutlibs\mylibrary.dll")  // set system ID
```

- LibID は、開発者によって選択されたインデックス番号です。この番号は、予約済みのシステム値 1024 ～ 2047 を除くゼロよりも大きい任意の整数です。
- ライブラリに既にインデックス番号 100 がロードされている場合、エントリを置き換える前にアンロードする必要があります。
- $ZF(-4,5) への呼び出しは、インデックス番号 100 をライブラリ・ファイル mylibrary.dll に関連付けます。

ライブラリ ID がシステム・インデックス・テーブルに定義されると、InterSystems IRIS の現在のインスタンス内のすべてのプロセスでグローバルに利用可能になります。

$ZF(-6) を使用した関数の呼び出し

この例では、前の例で作成されたシステム・インデックス・テーブルを使用します。$ZF(-6) を使用してライブラリをロードし、ライブラリ関数を呼び出して、ライブラリをアンロードします。このコードは、システム・インデックス内でライブラリ ID を定義したのと同じアプリケーションから呼び出す必要はありません。

ObjectScript

```
set LibID = 100   // library ID in system index table
set FuncID = 2    // second function in library ZFEntry table
set x = $ZF(-6,LibID, FuncID, "arg1")   // call function 2
set status = $ZF(-4,4,LibID)            // unload the library
```

- LibID は、システム・インデックス内に定義されているライブラリ ID です。このアプリケーションは、ライブラリ関数を使用するためにライブラリ名またはパスを知っている必要はありません。
- FuncID は、ライブラリ LibID の ZFEntry テーブルにリストされている 2 番目の関数の関数 ID です。開発者はライブラリ・コードへのアクセス権を持っていると見なされます ー $ZF(-6) インタフェースは、ライブラリ関数名を指定することによって、この番号を取得するための関数を持っていません。
- $ZF(-6) への呼び出しは、100 をライブラリ ID として指定し、2 を関数 ID として指定し、"arg1" を関数に渡す引数として指定します。この呼び出しは、コールアウト・ライブラリ mylibrary.dll がロードされていない場合にはロードし、ZFEntry テーブルにリストされている 2 番目の関数を呼び出します。
- $ZF(-4,4) への呼び出しは、ライブラリをアンロードします。$ZF(-6) によってロードされる各ライブラリは、プロセスが終了するまでまたは $ZF(-4,4) によってアンロードされるまで常駐したままになります。

5.4.1 ライブラリ関数のカプセル化での $ZF(-6) インタフェースの使用

$ZF(-5) インタフェースの例のように動作する (この章で前述した “システム ID によるライブラリへのアクセスでの $ZF(-5) の使用” を参照) $ZF(-6) インタフェースの例を記述するのは簡単ですが、これは $ZF(-6) を使用した利点を示しません。代わりに、このセクションでは、エンド・ユーザがコールアウト・ライブラリの内容または場所について知らずにまったく同じタスクを実行できるようにする ObjectScript クラスを提供します。

$ZF(-5) の例は、コールアウト・ライブラリ inputlibrary.dll および outputlibrary.dll から関数を呼び出して、いくつかの実験データを処理してグラフの描画に使用できる 2 次元の配列を生成しました。このセクションの例は、同じタスクを以下の ObjectScript コードを使用して実行します。

- クラス User.SystemIndex ー システム・インデックス・テーブル内でエントリを定義するために使用されるファイル名およびインデックス番号をカプセル化します。
- クラス User.GraphData ー 両方のライブラリから関数をカプセル化するメソッドを提供します。
- メソッド GetGraph() ー User.GraphData メソッドを呼び出すエンド・ユーザ・プログラムの一部です。このメソッドのコードは、$ZF(-5) の例とまったく同じタスクを実行しますが、$ZF 関数を直接呼び出しません。

User.SystemIndex クラスを使用すると、コールアウト・ライブラリを使用するアプリケーションがインデックス番号またはファイルの場所をハードコード化することなく、システム・インデックス・エントリを作成してアクセスすることができます。

ObjectScript クラス User.SystemIndex

Class Definition

```
Class User.SystemIndex Extends %Persistent
{
/// Defines system index table entries for the User.GraphData libraries
ClassMethod InitGraphData() As %Status
{
 // For each library, delete any existing system index entry and add a new one
    set sc = $ZF(-4,4,..#InputLibraryID)
    set sc = $ZF(-4,5,..#InputLibraryID,"c:\intersystems\iris\bin\inputlibrary.dll")
    set sc = $ZF(-4,4,..#OutputLibraryID)
    set sc = $ZF(-4,5,..#OutputLibraryID,"c:\intersystems\iris\bin\outputlibrary.dll")
    quit 0
}

Parameter InputLibraryID = 100;
Parameter OutputLibraryID = 200;
}
```

- InitGraphData() メソッドは、User.GraphData のライブラリをシステム・インデックス・テーブルに追加します。これは、InterSystems IRIS のインスタンスが開始されたときに自動的に呼び出され、ライブラリがインスタンス内のすべてのプロセスで利用可能になります。
- InputLibraryID および OutputLibraryID クラス・パラメータが利用可能になり、依存アプリケーションが (以下の例の User.GraphData の Init() メソッドによって示されているように) インデックス値をハードコード化する必要がなくなります。

User.GraphData クラスを使用すると、エンド・ユーザが実際のコールアウト・ライブラリについて何も知らなくてもライブラリ関数を呼び出すことができます。

ObjectScript クラス User.GraphData

Class Definition

```
Class User.GraphData Extends %Persistent
{
/// Gets library IDs and updates the system index table for both libraries.
Method Init() As %Status
{
    set InLibID = ##class(User.GraphDataIndex).%GetParameter("InputLibraryID")
    set OutLibID = ##class(User.GraphDataIndex).%GetParameter("OutputLibraryID")
    quit ##class(User.SystemIndex).InitGraphData()
}
Property InLibID As %Integer [Private];
Property OutLibID As %Integer [Private];

/// Calls function "FormatData" in library "inputlibrary.dll"
Method FormatData(rawdata As %Double) As %String
{
    quit $ZF(-6,..InLibID,1,rawdata)
}
/// Calls function "RefineData" in library "outputlibrary.dll"
Method RefineData(midvalue As %String) As %String
{
    quit $ZF(-6,..OutLibID,1,midvalue)
}
/// Calls function "PlotGraph" in library "outputlibrary.dll"
Method PlotGraph(datapoint As %String, xvalue As %Integer) As %String
{
    quit $ZF(-6,..OutLibID,2,datapoint,xvalue)
}
/// Unloads both libraries
Method Unload() As %String
{
    set sc = $ZF(-4,4,..InLibID)   // unload "inputlibrary.dll"
    set sc = $ZF(-4,4,..OutLibID)   // unload "outputlibrary.dll"
    quit 0
}
}
```

- Init() メソッドは、inputlibrary.dll および outputlibrary.dll のシステム・インデックス・エントリを設定または更新する User.SystemIndex からクラス・メソッドを呼び出します。また、ライブラリ ID の現在の値も取得します。このクラスの開発者は引き続き、コールアウト・ライブラリ・コードについて知っている必要がありますが、システム・インデックスへの将来の変更は透過的になります。
- メソッド FormatData()、RefineData()、および PlotGraph() はそれぞれ、1 つのライブラリ関数への呼び出しをカプセル化します。これらは無条件 $ZF 関数呼び出しのみを含むため、元の $ZF 呼び出しと同じ速さで実行するように最適化することができます。
- Unload() メソッドは、ライブラリのいずれか、または両方をアンロードします。

以下の例は、エンド・ユーザが User.GraphData 内でメソッドを使用する方法を示しています。GetGraph() メソッドは、コールアウト・ライブラリを使用して $ZF(-5) インタフェース例内の GraphSomeData() メソッドとまったく同じタスクを実行しますが (この章で前述している“システム ID によるライブラリへのアクセスでの $ZF(-5) の使用”を参照)、$ZF 関数を直接呼び出しません。

メソッド GetGraph()

```
Method GetGraph(loopsize As %Integer = 100000) As %Status
{
   // Get an instance of class GraphData and initialize the system index
   set graphlib = ##class(User.GraphData).%New()
   set sc = graphlib.Init()

   // call functions from both libraries repeatedly
   // each library is loaded automatically on first call
   for count=1:1:loopsize {
      set midvalue = graphlib.FormatData(^rawdata(count))
      set flatdata(count) = graphlib.RefineData(midvalue)
   }

   // plot results of the previous loop
   for count=1:1:loopsize {
      set x = graphlib.PlotGraph(flatdata(count),0)
      set y = graphlib.PlotGraph(flatdata(count),x)
      set ^graph(x,y) = flatdata(count)
   }

   //return after unloading all libraries loaded by $ZF(-6)
   set status = graphlib.Unload()
   quit 0
}
```

- User.GraphData クラスは graphlib としてインスタンス化され、Init() メソッドがシステム・インデックスを初期化するために呼び出されます。システム・インデックスは InterSystems IRIS のインスタンス内のすべてのプロセスに対して初期化する必要があるのは一度のみのため、このメソッドをここで呼び出す必要はありません。
- 最初のループは、間接的に $ZF(-6) を使用して、各ライブラリから関数を呼び出し、$ZF(-6) は、各ライブラリを最初に必要な際に自動的にロードします。ライブラリ inputlibrary.dll は、FormatData() への最初の呼び出しでロードされ、outputlibrary.dll は、RefineData() への最初の呼び出しでロードされます。
- 2 番目のループは、PlotGraph() を既にロードされているライブラリ outputlibrary.dll から呼び出します。
- Unload() への呼び出しは、両方のライブラリで間接的に $ZF(-4,4) を呼び出します。

5.4.2 テストでのプロセス・インデックスの使用

前に述べたように、プロセス・インデックス・テーブルは、システム・インデックス・テーブルの前に検索されるため、プロセス内でシステム全体の定義をオーバーライドするために使用できます。以下の例では、前のセクションで使用されたライブラリのうちの 1 つの新しいバージョンをテストするために使用されるプロセス・インデックスを作成します。

"inputlibrary.dll" の新しいバージョンをテストするためのプロセス・インデックスの使用

ObjectScript

```
// Initialize the system index and generate output from standard library
set testlib = ##class(User.GraphData).%New()
set sc = testlib.Init()
set sc = graphgen.GetGraph()   // get 100000 data items by default
merge testgraph1 = ^graph
kill ^graph

// create process index and test new library with same instance of testproc
set sc = $ZF(-4,4,100)   // unload current copy of inputlib
set sc = $ZF(-4,8)   // delete existing process index, if any
set sc = $ZF(-4,7,100, "c:\testfiles\newinputlibrary.dll")  // override system index
set sc = graphgen.GetGraph()
merge testgraph2 = ^graph

// Now compare testdata1 and testdata2
```

- 前の例のように、最初の 3 行で、このテスト・コードはシステム・インデックスを初期化してグラフを生成します。グラフは、inputlibrary.dll の標準バージョン (ID 値 100 を持つシステム・インデックス・エントリによって識別される) を使用してプロットされ、testgraph1 に保存されています。
- $ZF(-4,4) への呼び出しにより inputlibrary.dll がアンロードされます。これはシステム・インデックス・テーブル内でライブラリ ID 100 で識別されます。
- $ZF(-4,8) は、ライブラリ ID を指定せずに呼び出され、現在のプロセス・インデックス・テーブル内のすべてのエントリが削除されることを示します。
- $ZF(-4,7) への呼び出しは、テスト・ライブラリ newinputlibrary.dll に 100 をライブラリ ID として設定するプロセス・インデックス・テーブルにエントリを追加します。これにより、システム・インデックス内のその ID のエントリがオーバーライドされます。ライブラリ ID 100 は、inputlibrary.dll ではなく newinputlibrary.dll をポイントするようになります。
- GetGraph() が User.GraphData の同じインスタンスを使用して再び呼び出されます。標準バージョンの inputlibrary.dll がアンロードされる以外は何も変更されないため、GetGraph() は新しいバージョンのライブラリをロードして使用します。その後、テストではグラフ testgraph1 と testgraph2 が比較され、両方のバージョンで同じ結果を生成することが確認されます。

6

InterSystems $ZF コールアウトのクイック・リファレンス

$ZF() 関数は、1 つまたは 2 つの数値引数によって識別される従属関数のセットを提供します (例えば、$ZF(-100) 従属関数は外部プログラムまたはシステム・コマンドを実行し、$ZF(-4,1) 従属関数はコールアウト・ライブラリをロードします)。以下のリストは、特有の $ZF() 従属関数を識別する引数のみを示しています。これら関数の大半も、各関数の詳細エントリに記述されているように追加の引数を取ります。

関数の詳細な説明は、以下の見出しの下に構成されています。

- $ZF(-100) : プログラムまたはシステム・コマンドの実行
  - $ZF(-100) ー プログラムまたはシステム・コマンドを実行します。"ObjectScript リファレンス" の "$ZF(-100) (ObjectScript)" も参照してください。
- $ZF() : iriszf ライブラリの呼び出し
  - $ZF() (従属関数の引数なし) ー 現在のインスタンスの bin ディレクトリに配置された iriszf という名前のカスタム・コールアウト・ライブラリから関数の呼び出しを試行します。"ObjectScript リファレンス" の "$ZF() (ObjectScript)" も参照してください。
- $ZF(-3) : 名前による呼び出し
  - $ZF(-3) ー コールアウト・ライブラリをロードし、ライブラリ関数を呼び出します。"ObjectScript リファレンス" の "$ZF(-3) (ObjectScript)" も参照してください。
- $ZF(-5) : システム ID による呼び出し
  - $ZF(-5) ー システム定義の ID 番号によって参照されるコールアウト・ライブラリから関数を呼び出します。
    - $ZF(-4,1) ー 名前によって指定されるコールアウト・ライブラリをロードし、それに対する ID 番号を返します。
    - $ZF(-4,2) ー ID 番号によって指定されるコールアウト・ライブラリをアンロードするか、またはすべてのライブラリをアンロードします。
    - $ZF(-4,3) ー 指定されたライブラリ内の関数の ID 番号を返します。

    "ObjectScript リファレンス" の "$ZF(-5) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。
- $ZF(-6) : ユーザ・インデックスによる呼び出し
  - $ZF(-6) ー ユーザ指定のインデックス番号によって参照されるコールアウト・ライブラリから関数を呼び出します。
    - $ZF(-4,4) ー インデックス番号によって指定されるコールアウト・ライブラリをアンロードします。

- $ZF(-4,5) ー コールアウト・システム・インデックス・テーブル内にエントリを作成します。
- $ZF(-4,6) ー コールアウト・システム・インデックス・テーブル内のエントリを削除します。
- $ZF(-4,7) ー コールアウト・プロセス・インデックス・テーブル内にエントリを作成します。
- $ZF(-4,8) ー コールアウト・プロセス・インデックス・テーブル内のエントリを削除します。

"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

6.1 $ZF(-100) : プログラムまたはシステム・コマンドの実行

$ZF(-100) 関数は、外部プログラムまたはシステム・コマンドを実行するため、またはオペレーティング・システム・シェルを起動するために使用されます。これは、コールアウト・ライブラリなしで使用できる唯一の $ZF 関数です (詳細と例は、"$ZF(-100) を使用したプログラムまたはシステム・コマンドの実行" を参照してください)。

$ZF(-100)

プログラムまたはオペレーティング・システム・コマンドを実行します。

```
$ZF(-100, keyword_flags, program, arguments )
```

パラメータ :

- keyword_flags ー (オプション) 形式が `/keyword` のフラグのシーケンスで構成される文字列式。キーワードは大文字でも小文字でもかまいません。また、フラグ間に空白を使用できます。以下で説明しているように、入出力リダイレクトのキーワードの後ろには演算子とパス文字列 (`/keyword=path` または `/keyword+=path`) が続きます ("キーワードの指定" を参照)。
- program ー 実行するプログラムを指定します。これには、フル・パスまたは単なる名前を指定できます。この場合、オペレーティング・システムの通常の検索パス・ルールが適用されています。
- arguments ー (オプション) プログラムの引数のコンマ区切りリスト。可変個数のパラメータを arg... 構文で指定することもできます ("ObjectScript の使用法" の " 可変個数のパラメータ" を参照)。

これは、以下を返します。

以下の状態コードのいずれか :

- `-1` ー オペレーティング・システム・エラーが発生したので、詳細が SYSLOG に記録されます。
- `0` ー `/ASYNCH` が指定されている場合、プログラムが正常に開始されたことを示します。
- status ー `/ASYNCH` が指定されていない場合、status は、プログラムの終了時にプログラムから返される終了コードです (`0` または正の数値)。

キーワードの指定

以下のキーワードは、プログラムの実行とログへの記録を制御します。

- `/SHELL` ー プログラムをオペレーティング・システム・シェル内で呼び出す必要があることを示します。既定では、シェルは使用されません。
- `/ASYNC` ー プログラムを非同期で実行する必要があることを示します。これにより $ZF(-100) の呼び出しから、プログラムの完了を待たずに値が返されます。

- /LOGCMD ー プログラムのコマンド行が messages.log に記録されるようになります。これは、引数をプログラムで受け取られたとおりに表示する方法を提供するデバッグ・ツールです。

以下のキーワードとファイル指定子は、入出力リダイレクトを制御します。

- /STDIN=input-file
- /STDOUT=output-file または /STDOUT+=output-file
- /STDERR=error-file または /STDERR+=error-file

入出力リダイレクト・キーワードの後ろには、演算子 (= または +=) と、ファイル名またはファイル・パスが続きます。演算子の前後にスペースを入れることができます。標準の入力は、既存のファイルを指す必要があります。標準の出力および標準のエラー・ファイルは、存在しない場合は作成され、既に存在する場合は切り捨てられます。ファイルを作成または切り捨てるには = 演算子を使用し、既存のファイルに追加を行うには += 演算子を使用します。標準のエラーと標準の出力が同一のファイルに送信されるようにするには、両方のキーワードに同じファイルを指定します。

関連項目 :

詳細と例は、"$ZF(-100) を使用したプログラムまたはシステム・コマンドの実行" を参照してください。"ObjectScript リファレンス" の "$ZF(-100) (ObjectScript)" も参照してください。

6.2 $ZF() : iriszf ライブラリの呼び出し

$ZF() を負の数の引数なしで呼び出した場合 (例えば $ZF("myFunction",arg))、この関数は iriszf という名前のカスタム・コールアウト・ライブラリから関数の呼び出しを試行します ("$ZF() を使用した iriszf コールアウト・ライブラリへのアクセス" を参照)。

$ZF()

iriszf という名前のカスタム・コールアウト・ライブラリから関数の呼び出しを試行します。このライブラリを作成してインストールすると、その関数はすぐに $ZF() で利用できるようになり、ライブラリをロードしたり、ライブラリ識別子を指定したりする必要はありません。

```
retval = $ZF(func_name[, arg1[, ...argN]])
```

パラメータ :

- func_name ー ZFEntry テーブルで指定されているライブラリ関数の名前 ("ZFEntry テーブルの作成" を参照)。
- args ー (オプション) ライブラリ関数で必要とされる引数を含むコンマ区切りリスト。

これは、以下を返します。

- retval ー ライブラリ関数の出力値、またはライブラリ関数が出力値を設定しない場合は NULL。

関連項目 :

詳細と例は、"InterSystems コールアウト・ライブラリの作成" および "$ZF() を使用した iriszf コールアウト・ライブラリへのアクセス" を参照してください。"ObjectScript リファレンス" の "$ZF() (ObjectScript)" も参照してください。

6.3 $ZF(-3) : 名前による呼び出し

$ZF(-3) 関数と $ZF(-5) 関数を使用すると、アプリケーションで InterSystems コールアウト共有ライブラリをロードして実行時にライブラリ関数を呼び出すことができます。ライブラリ・パスとライブラリ関数名が呼び出し元のアプリケーションによって認識されている必要があります。$ZF(-3) は、ライブラリ名と関数名を引数として指定します。$ZF(-5) は、ライブラリと関数をシステム定義の ID 番号によって指定します。$ZF(-5) を使用する前に、ライブラリ名と関数名を引数として使用するユーティリティ関数 ($ZF(-4,1) ～ $ZF(-4,3)) を呼び出すことによって ID 番号を取得する必要があります。

$ZF(-3)

コールアウト・ライブラリをロードし、ライブラリ関数を実行します。一度にロードできるのは 1 つの $ZF(-3) ライブラリのみです。$ZF(-3) への呼び出しで、前の呼び出しとは異なるライブラリを指定する場合、前のライブラリはアンロードされ、置き換えられます。

```
retval = $ZF(-3, lib_name, func_name[, arg1[, ...argN]])
retval = $ZF(-3, lib_name, func_id[, arg1[, ...argN]])
```

パラメータ :

- lib_name ー ZFEntry テーブルに示されているコールアウト・ライブラリの名前 (“ZFEntry テーブルの作成” を参照)。ライブラリが既に前の $ZF(-3) の呼び出しによってロードされている場合は、空の文字列 ("") を使用して現在のライブラリを指定できます。
- func_name ー コールアウト・ライブラリ内で検索する関数の名前。
- func_id ー ZFEntry テーブル内のライブラリ関数のシーケンス番号。この番号が既知の場合は、アクセスを速くするために関数名の代わりに使用されます (エントリは 1 から連番で番号が付けられます)。
- args ー (オプション) ライブラリ関数で必要とされる引数を含むコンマ区切りリスト。

これは、以下を返します。

- retval ー ライブラリ関数の出力値、またはライブラリ関数が出力値を設定しない場合は NULL。

関連項目 :

詳細と例は、“単純なライブラリ関数呼び出しでの $ZF(-3) の使用” を参照してください。ZFEntry テーブル・シーケンス番号を取得する別の方法は、$ZF(-4,3) を参照してください。″ObjectScript リファレンス″ の ″$ZF(-3) (ObjectScript)″ も参照してください。

6.4 $ZF(-5) : システム ID による呼び出し

$ZF(-5) 関数を使用すると、アプリケーションで InterSystems コールアウト共有ライブラリをロードして実行時にライブラリ関数を呼び出すことができます。ライブラリ・パスとライブラリ関数名が呼び出し元のアプリケーションによって認識されている必要があります。ライブラリと関数は、システム定義 ID 番号によって指定されます。

ユーティリティ関数 $ZF(-4,1)、$ZF(-4,2)、$ZF(-4,3) は、$ZF(-5) でのみ使用されます。これらは、ライブラリ ID 番号を取得し、ライブラリをロードまたはアンロードするサービスを提供します。

$ZF(-5)

システム定義の ID 番号によって参照されるコールアウト・ライブラリから関数を呼び出します。

```
retval = $ZF(-5,lib_id,func_id,args)
```

パラメータ :

- lib_id ー $ZF(-4,1) によって指定されるコールアウト・ライブラリ ID 番号。
- func_id ー $ZF(-4,3) によって指定されるライブラリ関数 ID 番号。
- args ー (オプション) ライブラリ関数で必要とされる引数を含むコンマ区切りリスト。

これは、以下を返します。

- retval ー ライブラリ関数の出力値、またはライブラリ関数が出力値を設定しない場合は NULL。

関連項目 :

詳細と例は、"システム ID によるライブラリへのアクセスでの $ZF(-5) の使用" を参照してください。"ObjectScript リファレンス" の "$ZF(-5) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4, 1)

$ZF(-5) で使用されるユーティリティ関数。名前によって指定されるコールアウト・ライブラリをロードし、それに対する ID 番号を返します。

```
lib_id = $ZF(-4,1, lib_name)
```

パラメータ :

- lib_name ー ロードされるコールアウト・ライブラリの名前。

これは、以下を返します。

- lib_id ー lib_name を参照するために使用されるシステム定義識別子。

関連項目 :

詳細と例は、"システム ID によるライブラリへのアクセスでの $ZF(-5) の使用" を参照してください。"ObjectScript リファレンス" の "$ZF(-5) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4, 2)

$ZF(-5) で使用されるユーティリティ関数。ID 番号によって指定されるコールアウト・ライブラリをアンロードします。ID が指定されていない場合、$ZF(-4,1) または $ZF(6) のいずれかによってロードされたプロセスにあるすべてのライブラリがアンロードされます。$ZF(-3) によってロードされたライブラリはアンロードされません。

```
$ZF(-4,2[,lib_id])
```

パラメータ :

- lib_id ー $ZF(-4,1) によって返されるシステム定義識別子。指定されていない場合は、$ZF(-4,1) または $ZF(6) によってロードされたすべてのライブラリがアンロードされます。

関連項目 :

詳細と例は、"システム ID によるライブラリへのアクセスでの $ZF(-5) の使用" を参照してください。ライブラリ ID パラメータを指定せずに $ZF(-4,2) を使用する例は、"テストでのプロセス・インデックスの使用" も参照してください。"ObjectScript リファレンス" の "$ZF(-5) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4, 3)

$ZF(-5) で使用されるユーティリティ関数。指定されたライブラリ ID と関数名を持つ関数の ID 番号を返します。この番号は、実際には ZFEntry テーブル内の関数のシーケンス番号です (“ZFEntry テーブルの作成” を参照)。

```
func_id = $ZF(-4,3, lib_id, func_name)
```

パラメータ :

- lib_id ー $ZF(-4,1) によって返されるシステム定義のライブラリ識別子。
- func_name ー コールアウト・ライブラリ内で検索する関数の名前。

これは、以下を返します。

- func_id ー 指定されたライブラリ関数の返された ID 番号。

関連項目 :

詳細と例は、“システム ID によるライブラリへのアクセスでの $ZF(-5) の使用” を参照してください。″ObjectScript リファレンス″ の ″$ZF(-5) (ObjectScript)″ と ″$ZF(-4) (ObjectScript)″ も参照してください。

注釈　ユーティリティ関数 $ZF(-4, 4) から $ZF(-4, 8) については、次のセクション (“$ZF(-6) : ユーザ・インデックスによる呼び出し”) を参照してください。

6.5 $ZF(-6) : ユーザ・インデックスによる呼び出し

$ZF(-6) インタフェースは、ユーザ定義のインデックス・テーブルを介してコールアウト・ライブラリへのアクセスを可能にします。共有ライブラリ・ファイルの場所を認識していないアプリケーションでも使用可能です。

ユーティリティ関数 $ZF(-4, 4) から $ZF(-4, 8) は、$ZF(-6) によってのみ使用されます。これらは、ライブラリをアンロードし、インデックスを作成または維持するサービスを提供します。

$ZF(-6)

インデックス付きのコールアウト・ライブラリ内の関数を検索して実行します。

```
retval = $ZF(-6,lib_index,func_id,args)
```

パラメータ :

- lib_index ー コールアウト・ライブラリに対するユーザ指定のインデックス ($ZF(-4,5) または $ZF(-4,7) によって作成)。
- func_id ー (オプション) ライブラリの ZFEntry テーブル内の関数インデックスである、コールアウト・ライブラリ内の関数の ID 番号。これを省略した場合は、呼び出しによって lib_index の妥当性を検証し、ライブラリをロードして、完全なライブラリ・ファイル名を返します。
- args ー (オプション) ライブラリ関数で必要とされる引数を含むコンマ区切りリスト。

これは、以下を返します。

- retval ー ライブラリ関数の出力値、またはライブラリ関数が出力値を設定しない場合は NULL。

関連項目 :

詳細と例は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” を参照してください。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

注釈　ユーティリティ関数 $ZF(-4, 1)、$ZF(-4, 2)、$ZF(-4, 3) については、前のセクション (“$ZF(-5) : システム ID による呼び出し”) を参照してください。

$ZF(-4, 4)

$ZF(-6) で使用されるユーティリティ関数。インデックス番号によって指定されるコールアウト・ライブラリをアンロードします。

```
$ZF(-4,4,lib_index)
```

パラメータ :

- lib_index ー ユーザ指定のコールアウト・ライブラリ・インデックス番号 ($ZF(-4,5) または $ZF(-4,7) によって作成)。

関連項目 :

詳細と例は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” を参照してください。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4, 5)

$ZF(-6) で使用されるユーティリティ関数。コールアウト・システム・インデックス・テーブル内にエントリを作成します。

```
$ZF(-4,5,lib_index,lib_name)
```

パラメータ :

- lib_index ー コールアウト・ライブラリを参照するために使用される一意のユーザ指定番号。
- lib_name ー インデックスが付けられるコールアウト・ライブラリの名前。

関連項目 :

“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” の詳細と例。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4, 6)

$ZF(-6) で使用されるユーティリティ関数。コールアウト・システム・インデックス・テーブル内のエントリを削除します。

```
$ZF(-4,6,lib_index)
```

パラメータ :

- lib_index ー $ZF(-4,5) への呼び出しによって前に定義されたインデックス番号。この引数は必須です (省略可能な $ZF(-4,8) とは異なります)。

関連項目 :

詳細と例は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” を参照してください。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4,7)

$ZF(-6) で使用されるユーティリティ関数。コールアウト・プロセス・インデックス・テーブル内にエントリを作成します。

```
$ZF(-4,7,lib_index,lib_name)
```

パラメータ：

- lib_index ー コールアウト・ライブラリを参照するために使用される一意のユーザ指定番号。
- lib_name ー インデックスが付けられるコールアウト・ライブラリの名前。

関連項目：

詳細と例は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” を参照してください。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。

$ZF(-4,8)

$ZF(-6) で使用されるユーティリティ関数。コールアウト・プロセス・インデックス・テーブル内からエントリを削除します。インデックス番号が指定されていない場合、すべてのインデックス・エントリが削除されます。

```
$ZF(-4,8,lib_index)
```

パラメータ：

- lib_index ー (オプション) $ZF(-4,7) への呼び出しによって前に定義されたインデックス番号。指定されていない場合、すべてのインデックス・エントリが削除されます。

関連項目：

詳細と例は、“ユーザ・インデックスによるライブラリへのアクセスでの $ZF(-6) の使用” を参照してください。"ObjectScript リファレンス" の "$ZF(-6) (ObjectScript)" と "$ZF(-4) (ObjectScript)" も参照してください。