



サーバ側プログラミングの入門ガイド

Version 2023.1
2024-01-02

サーバ側プログラミングの入門ガイド

InterSystems IRIS Data Platform Version 2023.1 2024-01-02

Copyright © 2024 InterSystems Corporation

All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, および TrakCare は、InterSystems Corporation の登録商標です。HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, および InterSystems TotalView™ For Asset Management は、InterSystems Corporation の商標です。TrakCare は、オーストラリアおよび EU における登録商標です。

ここで使われている他の全てのブランドまたは製品名は、各社および各組織の商標または登録商標です。

このドキュメントは、インターシステムズ社(住所: One Memorial Drive, Cambridge, MA 02142)あるいはその子会社が所有する企業秘密および秘密情報を含んでおり、インターシステムズ社の製品を稼動および維持するためにのみ提供される。この発行物のいかなる部分も他の目的のために使用してはならない。また、インターシステムズ社の書面による事前の同意がない限り、本発行物を、いかなる形式、いかなる手段で、その全てまたは一部を、再発行、複製、開示、送付、検索可能なシステムへの保存、あるいは人またはコンピュータ言語への翻訳はしてはならない。

かかるプログラムと関連ドキュメントについて書かれているインターシステムズ社の標準ライセンス契約に記載されている範囲を除き、ここに記載された本ドキュメントとソフトウェアプログラムの複製、使用、廃棄は禁じられている。インターシステムズ社は、ソフトウェアライセンス契約に記載されている事項以外にかかるソフトウェアプログラムに関する説明と保証をするものではない。さらに、かかるソフトウェアに関する、あるいはかかるソフトウェアの使用から起こるいかなる損失、損害に対するインターシステムズ社の責任は、ソフトウェアライセンス契約にある事項に制限される。

前述は、そのコンピュータソフトウェアの使用およびそれによって起こるインターシステムズ社の責任の範囲、制限に関する一般的な概略である。完全な参照情報は、インターシステムズ社の標準ライセンス契約に記載され、そのコピーは要望によって入手することができる。

インターシステムズ社は、本ドキュメントにある誤りに対する責任を放棄する。また、インターシステムズ社は、独自の裁量にて事前通知なしに、本ドキュメントに記載された製品および実行に対する代替と変更を行う権利を有する。

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 InterSystems IRIS プログラミングの概要	1
1.1 はじめに	1
1.2 クラス	1
1.3 ルーチン	3
1.4 クラスとルーチンを一緒に使用方法	3
1.5 グローバルの概要	4
1.5.1 データへのアクセス方法	4
1.6 InterSystems SQL	4
1.6.1 ObjectScript からの SQL の使用	4
1.6.2 Python からの SQL の使用	5
1.7 マクロ	5
1.8 インクルード・ファイル	6
1.9 これらのコード要素がどのように連携しているか	6
2 ObjectScript の詳細	9
2.1 サンプル・クラス	9
2.2 サンプル・ルーチン	10
2.3 変数	12
2.3.1 変数名	12
2.3.2 変数タイプ	13
2.3.3 可変長	13
2.3.4 変数の存在	13
2.4 変数の可用性と範囲	14
2.4.1 変数の範囲の概要	14
2.5 多次元配列	15
2.5.1 基本	15
2.5.2 構造のバリエーション	16
2.5.3 使用上の注意	16
2.6 値または参照による変数渡し	17
2.7 演算子	18
2.7.1 よく知られている演算子	19
2.7.2 あまり知られていない演算子	19
2.8 コマンド	20
2.8.1 よく知られているコマンド	20
2.8.2 多次元配列で使用するコマンド	21
2.9 特殊変数	21
2.9.1 \$SYSTEM 特殊変数	22
2.10 ロックと並行処理の制御	23
2.10.1 基本	23
2.10.2 ロック・テーブル	23
2.10.3 ロックと配列	24
2.10.4 ロック・タイプの概要	24
2.11 システム関数	25
2.11.1 値の選択	25
2.11.2 存在の関数	26
2.11.3 リスト関数	26
2.11.4 文字列関数 (1)	26
2.11.5 多次元配列の操作	27

2.11.6 文字値	27
2.12 日付と時刻の値	27
2.12.1 ローカル時刻	28
2.12.2 UTC 時刻	28
2.12.3 日付・時刻の変換	28
2.12.4 \$H 形式の詳細	28
2.13 マクロとインクルード・ファイルの使用	29
2.13.1 マクロ	29
2.13.2 インクルード・ファイル	29
2.14 ルーチンの使用	30
2.14.1 プロシージャ、関数、およびサブルーチン	30
2.14.2 ルーチンの実行	32
2.14.3 NEW コマンド	33
2.15 潜在的な落とし穴	33
2.16 詳細	35
3 クラス	37
3.1 クラス名とパッケージ	37
3.2 クラス定義の基本的なコンテンツ	37
3.3 クラス・メソッド呼び出しのショートカット	40
3.4 クラス・パラメータ	40
3.5 プロパティ	41
3.5.1 プロパティ・キーワードの指定	42
3.6 データ型に基づくプロパティ	43
3.6.1 データ型クラス	43
3.6.2 データ型クラスのパラメータのオーバーライド	44
3.6.3 他のプロパティ・メソッドの使用法	44
3.7 メソッド	45
3.7.1 Method キーワードの指定	45
3.7.2 他のクラス・メンバの参照	46
3.7.3 他のクラスのメソッドの参照	47
3.7.4 現在のインスタンスの参照	49
3.7.5 メソッド引数	49
3.8 メソッド・ジェネレータ	51
3.9 クラス・クエリ	51
3.10 XData ブロック	52
3.11 クラス定義におけるマクロとインクルード・ファイル	52
3.12 InterSystems IRIS における継承規則	52
3.12.1 継承順序	53
3.12.2 プライマリ・スーパークラス	53
3.12.3 最も適切なタイプのクラス	53
3.12.4 メソッドのオーバーライド	53
3.13 詳細	54
4 オブジェクト	55
4.1 InterSystems IRIS オブジェクト・クラスの概要	55
4.2 オブジェクト・クラスの基本機能	56
4.3 OREF	58
4.4 ストリーム・インタフェース・クラス	59
4.4.1 ストリーム・クラス	59
4.4.2 例	60
4.5 コレクション・クラス	60

4.5.1 スタンドアロン・オブジェクトとして使用するリストおよび配列クラス	60
4.5.2 プロパティとしてのリストおよび配列	61
4.6 便利な ObjectScript 関数	62
4.7 詳細	63
5 永続オブジェクトと InterSystems IRIS SQL	65
5.1 はじめに	65
5.2 InterSystems SQL	65
5.2.1 ObjectScript からの SQL の使用	66
5.2.2 Python からの SQL の使用	66
5.2.3 SQL へのオブジェクト拡張	67
5.3 永続クラスに対する特別なオプション	67
5.4 永続クラスの SQL プロジェクション	68
5.4.1 オブジェクト SQL プロジェクションのデモ	68
5.4.2 オブジェクト SQL プロジェクションの基本事項	69
5.4.3 クラスとエクステンツ	70
5.5 オブジェクト ID	70
5.5.1 ID はどのように決まるか	71
5.5.2 ID へのアクセス	71
5.6 ストレージ	72
5.6.1 ストレージ定義の確認	72
5.6.2 永続クラスによって使用されるグローバル	73
5.6.3 格納されるオブジェクトの既定の構造	73
5.6.4 メモ	73
5.7 永続クラスおよびテーブルを作成するためのオプション	74
5.8 データへのアクセス	74
5.9 格納されているデータの確認	75
5.10 InterSystems SQL に対して生成されたコードのストレージ	76
5.11 詳細	77
6 ネームスペースとデータベース	79
6.1 ネームスペースとデータベースの概要	79
6.1.1 ロック、グローバル、およびネームスペース	80
6.2 データベースの基本事項	80
6.2.1 データベース構成	80
6.2.2 データベースの特徴	81
6.2.3 データベースの移植性	81
6.3 システム提供データベース	82
6.4 %SYS ネームスペース	83
6.5 IRISSYS データベースおよびカスタム項目	83
6.6 ネームスペースで何にアクセス可能か	83
6.6.1 ネームスペースのシステム・グローバル	84
6.7 システム・ディレクトリ	84
6.8 詳細	85
7 InterSystems IRIS セキュリティ	87
7.1 概要	87
7.1.1 InterSystems IRIS 内のセキュリティ要素	87
7.1.2 InterSystems IRIS との安全な接続	88
7.2 InterSystems IRIS アプリケーション	88
7.3 InterSystems 認証モデル	88
8 ローカライズのサポート	91

8.1 概要	91
8.2 InterSystems IRIS のロケールと各国言語のサポート	91
8.3 既定の入出力テーブル	92
8.4 ファイルと文字エンコード	93
8.5 文字の手動変換	93
9 サーバ構成オプション	95
9.1 InterSystems SQL の設定	95
9.1.1 クエリキャッシュのソースを保持	95
9.1.2 デフォルトスキーマ	95
9.1.3 区切り識別子のサポート	96
9.2 IPv6 アドレスの使用	96
9.3 プログラムによるサーバの構成	96
9.4 詳細	96
10 効果的なスキル	97
10.1 データベースの定義	97
10.2 ネームスペースの定義	97
10.3 グローバルのマッピング	98
10.4 ルーチンのマッピング	99
10.5 パッケージのマッピング	100
10.6 テスト・データの生成	101
10.6.1 %Populate の拡張	101
10.6.2 %Populate および %PopulateUtils のメソッドの使用	102
10.7 格納したデータの削除	102
10.8 ストレージのリセット	103
10.9 テーブルの参照	103
10.10 SQL クエリの実行	104
10.11 オブジェクト・プロパティの検証	105
10.12 グローバルの表示	106
10.13 クエリのテストとクエリ・プランの表示	107
10.14 クエリ・キャッシュの表示	107
10.15 インデックスの構築	108
10.16 テーブルのチューニング機能の使用	109
10.17 1 つのデータベースから別のデータベースへのデータの移動	109
付録A: Caché の様々な構文	111
A.1 “単語“ の中の非英数字	111
A.2 .(ピリオド 1 つ)	113
A.3 ..(2 つのピリオド)	114
A.4 # (シャープ記号)	114
A.5 ドル記号 (\$)	115
A.6 パーセント記号 (%)	116
A.7 キャレット (^)	117
A.8 その他の形式	118
付録B: 識別子のルールとガイドライン	121
B.1 ネームスペース	121
B.1.1 回避する必要があるネームスペース名	121
B.2 データベース	121
B.2.1 回避する必要があるデータベース名	121
B.3 ローカル変数	122
B.3.1 回避する必要があるローカル変数名	122

B.4 グローバル変数	122
B.4.1 回避する必要があるグローバル変数名	123
B.5 ルーチンとラベル	124
B.5.1 ユーザが使用するために予約されているルーチン名	125
B.6 クラス	125
B.6.1 回避する必要があるクラス名	126
B.7 クラス・メンバ	126
B.7.1 回避する必要があるメンバ名	127
B.8 IRISSYS データベースおよびカスタム項目	127
付録C: 一般的なシステム制限	129
C.1 文字列長の制限	129
C.2 クラスの制限	129
C.3 クラスおよびルーチンの制限	130
C.4 その他のプログラミング制限	132
付録D: インターシステムズ・アプリケーションでの数値の計算	135
D.1 数値の表現	135
D.1.1 10 進形式	135
D.1.2 \$DOUBLE 形式	136
D.1.3 SQL 表現	137
D.2 数値形式の選択	137
D.3 数値表現の変換	137
D.3.1 文字列	138
D.3.2 10 進数から \$DOUBLE への変換	138
D.3.3 \$DOUBLE から 10 進数への変換	139
D.3.4 10 進数から文字列への変換	139
D.4 数値を含む演算	140
D.4.1 算術演算子	140
D.4.2 比較演算子	140
D.4.3 ブーリアン演算	141
D.5 値の正確な表現	141
D.6 関連項目	143

1

InterSystems IRIS プログラミングの概要

このページでは、InterSystems IRIS® サーバ側プログラムで利用できる言語要素の概要を示します。

1.1 はじめに

InterSystems IRIS は、ObjectScript と呼ばれる汎用プログラミング言語と Python のサポートが組み込まれた高性能のマルチモデル・データ・プラットフォームです。

InterSystems IRIS では、多重プロセスがサポートされ、並列処理の制御が可能です。各プロセスは、データに直接アクセスします。

InterSystems IRIS では、自身の嗜好に合わせて、クラス、ルーチン、またはそれらの組み合わせを記述できます。どの場合でも、格納するデータは最終的に**グローバル**という構造に収められます。InterSystems IRIS プログラミングには、以下の機能があります。

- ・ クラスとルーチンは相互に交換して使用できます。
- ・ クラスとルーチンはお互いを呼び出すことができます。
- ・ クラスには、オブジェクト指向の機能が用意されています。
- ・ データベース・ストレージは、ObjectScript と Python の両方に統合されます。
- ・ クラスは、プログラミングが簡素化されるようにデータを永続化できます。永続クラスを使用する場合、データは、オブジェクト、SQL テーブル、およびグローバルとして同時に使用できます。
- ・ グローバルには、クラスとルーチンのどちらからでも直接アクセスできます。つまり、完全に目的どおりにデータの格納およびデータへのアクセスを行える柔軟性を備えています。

ユーザのニーズに合ったアプローチを選択できます。

1.2 クラス

InterSystems IRIS ではクラスがサポートされます。システム・クラスを使用することも、独自のクラスを定義することもできます。

InterSystems IRIS では、プロパティ、メソッド、パラメータ (他のクラス言語では定数と呼ぶ) など、よく知られたクラス要素をクラスに含めることができます。また、トリガ、クエリ、インデックスなど、クラスでは通常定義されない項目を含めることもできます。

InterSystems IRIS クラス定義では、クラス定義言語 (CDL) を使用して、クラスとそのメンバ (プロパティ、メソッド、パラメータなど) を指定します。メソッド内部の実行可能コードの記述には、Python または ObjectScript を使用できます。メソッドごとに、Language キーワードを使用して、そのメソッドの記述に使用する言語を指定します。以下に例を示します。

クラス定義を以下に示します。

Class/ObjectScript

```
Class Sample.Employee Extends %Persistent
{
    /// The employee's name.
    Property Name As %String(MAXLEN = 50);

    /// The employee's job title.
    Property Title As %String(MAXLEN = 50);

    /// The employee's current salary.
    Property Salary As %Integer(MAXVAL = 100000, MINVAL = 0);

    /// This method prints employee information using ObjectScript.
    Method PrintEmployee() [ Language = objectscript ]
    {
        Write !,"Name: ", ..Name, " Title: ", ..Title
    }
}
```

Class/Python

```
Class Sample.Employee Extends %Persistent
{
    /// The employee's name.
    Property Name As %String(MAXLEN = 50);

    /// The employee's job title.
    Property Title As %String(MAXLEN = 50);

    /// The employee's current salary.
    Property Salary As %Integer(MAXVAL = 100000, MINVAL = 0);

    /// This method prints employee information using Embedded Python.
    Method PrintEmployee() [ Language = python ]
    {
        print("\nName:", self.Name, "Title:", self.Title)
    }
}
```

メソッドで使用する言語を指定しない場合、コンパイラはメソッドが ObjectScript で記述されていると想定します。

[InterSystems IRIS の各種クラス](#)および [InterSystems IRIS の各種永続クラス](#)固有の機能については別途説明します。

1.3 ルーチン

InterSystems IRIS でルーチンを作成する場合は [ObjectScript](#) を使用します。以下に示すのは、ObjectScript で記述されたルーチンの一部です。

```
SET text = ""
FOR i=1:5:$LISTLENGTH(attrs)
{
    IF ($ZCONVERT($LIST(attrs, (i + 1)), "U") = "XREFLABEL")
    {
        SET text = $LIST(attrs, (i + 4))
        QUIT
    }
}

IF (text = "")
{
    QUIT $$$ERROR($$$GeneralError,$$$T("Missing xreflabel value"))
}
```

1.4 クラスとルーチンを一緒に使用する方法

InterSystems IRIS では、クラスをルーチン内から使用できます。例えば、以下はルーチンの一部を示しており、ここでは `Sample.Employee` クラスを参照しています。

ObjectScript

```
//get details of random employee and print them
showemployee() public {
    set rand=$RANDOM(10)+1 ; rand is an integer in the range 1-10
    write "Your random number: " _rand
    set employee=##class(Sample.Employee).%OpenId(rand)
    do employee.PrintEmployee()
    write !,"This employee's salary: " _employee.Salary
}
```

同様に、メソッドはルーチン内のラベルを呼び出すことができます。例えば、以下はルーチン `employeeutils` 内のラベル `ComputeRaise` を呼び出します。

Method/ObjectScript

```
Method RaiseSalary() As %Numeric
{
    set newsalary=$$ComputeRaise^employeeutils(..Salary)
    return newsalary
}
```

Method/Python

```
Method RaiseSalary() as %Numeric [ Language = python ]
{
    import iris
    newsalary=iris.routine("ComputeRaise^employeeutils", self.Salary)
    return newsalary
}
```

1.5 グローバルの概要

InterSystems IRIS では、他のプログラミング言語にはない特殊な変数がサポートされています。それがグローバル変数です (通常、単にグローバルと呼ばれます)。InterSystems IRIS では、グローバルという用語は、そのデータが、このデータベースにアクセスするすべてのプロセスで使用可能であることを示します。その使用は他のプログラミング言語とは異なり、他のプログラミング言語では、グローバルは “そのモジュールのすべてのコードで使用可能であること” を意味します。

グローバルのコンテンツは、InterSystems IRIS データベースに保存されます。ここでは、以下の点のみを理解しておくことが重要です。

- ・ グローバルは、データベースに自動的に格納されます。グローバル変数のノードに値を割り当てると、そのデータは直ちにデータベースに書き込まれます。
- ・ グローバルのコンテンツは、プログラムによって、または管理ポータルを使用して表示できます。

1.5.1 データへのアクセス方法

InterSystems IRIS では、データベースにグローバル以外は何も含まれていません。このドキュメントで後述するように、コードもグローバルに格納されます。最も低いレベルでは、データへのアクセスはすべて、直接グローバル・アクセスを介して、つまり、グローバルを直接操作するコマンドおよび関数を使用して行われます。

このドキュメントで後述する永続クラスを使用する場合は、以下の方法で、格納されるデータを作成、変更、および削除できます。

- ・ ObjectScript で、%New()、%Save()、%Open()、%Delete() などのメソッドを使用する。
- ・ Python で、_New()、_Save()、_Open()、_Delete() などのメソッドを使用する。
- ・ ObjectScript で、直接グローバル・アクセスを使用する。
- ・ Python で、gref() メソッドを使用して直接グローバル・アクセスを提供する。
- ・ InterSystems SQL を使用

内部では、システムは常に、直接グローバル・アクセスを使用します。

1.6 InterSystems SQL

InterSystems IRIS は、InterSystems SQL と呼ばれる SQL の実装を提供します。InterSystems SQL は、メソッド内およびルーチン内で使用できます。

1.6.1 ObjectScript からの SQL の使用

以下のいずれかまたは両方の方法を使用して、ObjectScript から SQL を実行できます。

- ・ ダイナミック SQL (%SQL.Statement クラスおよび %SQL.StatementResult クラス)。以下に例を示します。

ObjectScript

```
SET myquery = "SELECT TOP 5 Name, Title FROM Sample.Employee ORDER BY Salary"
SET tStatement = ##class(%SQL.Statement).%New()
SET tStatus = tStatement.%Prepare(myquery)
SET rset = tStatement.%Execute()
DO rset.%Display()
WRITE !,"End of data"
```

ダイナミック SQL は、ObjectScript のメソッドおよびルーチンで使用できます。

- ・ 埋め込み SQL。以下に例を示します。

ObjectScript

```
&sql(SELECT COUNT(*) INTO :myvar FROM Sample.Employee)
  IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg QUIT}
  ELSEIF SQLCODE=100 {WRITE "Query returns no results" QUIT}
WRITE myvar
```

最初の行は埋め込み SQL であり、InterSystems SQL クエリを実行し、`myvar` と呼ばれるホスト変数に値を書き込みます。

次の行は通常の ObjectScript であり、変数 `myvar` の値を書き込むのみです。

埋め込み SQL は、ObjectScript のメソッドおよびルーチンで使用できます。

1.6.2 Python からの SQL の使用

Python からの SQL の使用は、ObjectScript からのダイナミック SQL の使用と同様です。以下のいずれかまたは両方の方法を使用して、Python から SQL を実行できます。

- ・ SQL クエリを直接実行できます。以下に例を示します。

Python

```
import iris
rset = iris.sql.exec("SELECT * FROM Sample.Employee ORDER BY Salary")
for row in rset:
    print(row)
```

2 行目で InterSystems SQL クエリを実行し、変数 `rset` に格納された結果セットを返します。

- ・ 最初に SQL クエリを準備してから実行することもできます。以下に例を示します。

Python

```
import iris
statement = iris.sql.prepare("SELECT * FROM Sample.Employee ORDER BY Salary")
rset = statement.execute()
for row in rset:
    print(row)
```

この例では、2 行目で SQL クエリを返します。この SQL が 3 行目で実行され、結果セットが返されます。

Python ターミナルまたは Python メソッドでこれらのいずれかの方法を使用して、SQL クエリを実行できます。

1.7 マクロ

ObjectScript では、置換を定義するマクロもサポートします。その定義は、値、あるコード行の全体、(##continue 指示文も含む) 複数の行のいずれかです。マクロは、整合性を確保するために使用します。以下はその例です。

ObjectScript

```
#define StringMacro "Hello, World!"  
  
write $$$StringMacro
```

マクロの使用の詳細は、“[マクロとインクルード・ファイルの使用](#)”を参照してください。

1.8 インクルード・ファイル

ルーチンの中でマクロを定義し、それを同じルーチンの中で後で使用できます。より一般的には、それらは中央の場所で定義します。このためには、インクルード・ファイルを作成して使用します。インクルード・ファイルは、マクロを定義し、他のインクルード・ファイルをインクルードできます。

インクルード・ファイルの使用法の詳細は、“[マクロとインクルード・ファイルの使用](#)”を参照してください。

1.9 これらのコード要素がどのように連携しているか

このページで紹介するコード要素が InterSystems IRIS でどのように使用されるのかを理解しておく役立ちます。

ObjectScript、Python、InterSystems SQL、クラス定義、マクロ、ルーチンなどを組み合わせて使用できるのは、記述したコードを InterSystems IRIS が直接使用しているのではないからです。代わりに、コードのコンパイル時に、システムが使用する下位レベルのコードが生成されます。これは、ObjectScript の場合は ObjectScript エンジンで使用される OBJ コードであり、Python の場合は Python エンジンで使用される PYC コードです。

複数のステップがあります。ステップを詳細に理解しておく必要はありませんが、以下の点を覚えておくことをお勧めします。

- ・ クラス・コンパイラは、Python メソッド以外のすべての要素について、クラス定義および ObjectScript コードを処理して INT コードを生成します。Python コードは PY コードに処理されます。

場合によっては、コンパイラは、ユーザが編集できない追加クラスを生成および保存します。これは、例えば、Web サービスおよび Web クライアントを定義するクラスをコンパイルしたときに行われます。

クラス・コンパイラは、各クラスのクラス記述子も生成します。これは、システム・コードが実行時に使用します。

- ・ ObjectScript コードの場合、プリプロセッサ (マクロ・プリプロセッサまたは MPP と呼ばれることもあります) は、インクルード・ファイルを使用し、マクロを置換します。また、ルーチン内の埋め込み SQL も処理します。

これらの変更は、一時的な作業領域で行われ、コードが変更されることはありません。

- ・ ルーチンの INT コードは追加のコンパイラによって作成されます。
- ・ INT コードおよび PY コードは中間層であり、この層では、データへのアクセスは直接グローバル・アクセスを使用して処理されます。これは人間が読めるコードです。
- ・ INT コードは OBJ コードの生成に使用され、PY コードは PYC コードの生成に使用されます。このコードは、InterSystems IRIS 仮想マシンによって使用されます。コードを OBJ コードや PYC コードにコンパイルしたら、INT ルーチンや PY ルーチンはコードの実行に不要となります。
- ・ クラスをコンパイルしたら、それらを配置モードにすることができます。InterSystems IRIS には、指定クラスの間コードと内部クラスを削除するユーティリティがあり、このユーティリティはアプリケーションを配置するときに使用できます。

InterSystems IRIS システム・クラスを調べると、クラスによっては配置モードになっているため、見えないことがあります。

注釈 クラス定義およびルーチンはすべて、生成済みコードとして同じ InterSystems IRIS データベースに格納されます。これにより、コードの管理が容易になります。InterSystems IRIS には一連の堅牢なソース・コントロール・フックが用意されており、インターシステムズの開発者はそれらを長年にわたって使用してきました。これらのフックはユーザも使用できます。

2

ObjectScript の詳細

このページでは、ObjectScript 言語の概要を説明すると共に、ObjectScript でプログラミングする必要のあるユーザや、他のユーザが記述したコードを理解する必要があるユーザ向けに有用な背景情報を紹介します。

メソッドとルーチンはどちらも ObjectScript で記述できますが、現在のコードの大半はメソッドを使用して記述します。メソッドはクラス内に収められます。これにより、類似メソッドをグループにまとめたり、クラス・リファレンスのドキュメントを自動生成したり、InterSystems IRIS のオブジェクト指向機能を使用したりできます。

ルーチンが重要ではないわけではありません。数多くの有用なシステム・ユーティリティがルーチンとして記述されます。ルーチンはクラスのコンパイル時に生成されます。

2.1 サンプル・クラス

以下に、ObjectScript で記述されたメソッドが含まれる `User.DemoClass` という名前のサンプル・クラスを示します。この例は、いくつかの一般的な ObjectScript コマンド、演算子、および関数と、メソッド内でのコードの構成方法を示しています。

Class Definition

```
Class User.DemoClass
{

    /// Generate a random number.
    /// This method can be called from outside the class.
    ClassMethod Random() [ Language = objectscript ]
    {
        set rand=$RANDOM(10)+1          ; rand is an integer in the range 1-10
        write "Your random number: "_rand
        set name=..GetNumberName(rand)
        write !, "Name of this number: "_name
    }

    /// Input a number.
    /// This method can be called from outside the class.
    ClassMethod Input() [ Language = objectscript ]
    {
        read "Enter a number from 1 to 10: ", input
        set name=..GetNumberName(input)
        write !, "Name of this number: "_name
    }

    /// Given an number, return the name.
    /// This method can be called only from within this class.
    ClassMethod GetNumberName(number As %Integer) As %Integer [ Language = objectscript, Private ]
    {
        set name=$CASE(number,1:"one",2:"two",3:"three",
            4:"four",5:"five",6:"six",7:"seven",8:"eight",
            9:"nine",10:"ten",:"other")
        quit name
    }
}
```

```

/// Write some interesting values.
/// This method can be called from outside the class.
ClassMethod Interesting() [ Language = objectscript ]
{
    write "Today's date: " _$ZDATE($HOROLOG,3)
    write !,"Your installed version: " _$ZVERSION
    write !,"Your username: " _$USERNAME
    write !,"Your security roles: " _$ROLES
}

```

以下の点に注意してください。

- Random() メソッドおよび Input() メソッドは、GetNumberName() メソッド (このクラスに対してプライベートであり、クラスの外部からは呼び出せない) を呼び出します。
- WRITE、QUIT、SET、および READ はコマンドです。この言語には、ほかに変数を削除するコマンド、プログラム・フローを制御するコマンド、入出力デバイスを制御するコマンド、トランザクションを管理するコマンド (ネスト可能) などがあります。

コマンドの名前では大文字小文字は区別されませんが、規約によって実行されるテキストではすべて大文字で表記します。

- このサンプルには 2 つの演算子が含まれています。プラス記号 (+) は加算を、アンダースコア () は文字列連結を実行します。

ObjectScript には、通常の演算子以外に他の言語にはないいくつかの特別な演算子があります。

- \$RANDOM、\$CASE、および \$ZDATE は関数です。
この言語は、文字列処理、多くの種類の変換、フォーマット処理、算術演算などのための関数を提供します。
- \$HOROLOG、\$ZVERSION、\$USERNAME、および \$ROLES は、システム変数 (InterSystems IRIS では特殊変数と呼ぶ) です。大部分の特殊変数には、InterSystems IRIS 操作環境の状況、現在の処理状態などに対応した値が格納されます。
- ObjectScript は、コメント行、ブロック・コメント、文の終わりのコメントをサポートします。

デモンストレーションとして、このクラスのメソッドをターミナルで実行できます。これらの例では、TESTNAMESPACE> はターミナルに表示されるプロンプトです。同じ行の、プロンプトの後にあるテキストは、入力されたコマンドです。その後の行は、応答としてシステムによってターミナルに書き込まれた値を示しています。

```

TESTNAMESPACE>do ##class(User.DemoClass).Input()
Enter a number from 1 to 10: 7
Name of this number: seven
TESTNAMESPACE>do ##class(User.DemoClass).Interesting()
Today's date: 2021-07-15
Your installed version: IRIS for Windows (x86-64) 2019.3 (Build 310U) Mon Oct 21 2019 13:48:58 EDT
Your username: SuperUser
Your security roles: %All
TESTNAMESPACE>

```

2.2 サンプル・ルーチン

以下に、ObjectScript で記述された demoroutine という名前のサンプル・ルーチンを示します。このルーチンには、前述のセクションのサンプル・クラスで示したメソッドとまったく同じ処理を行うプロシージャが含まれます。

ObjectScript

```

; this is demoroutine
write "Use one of the following entry points:"
write !,"random"
write !,"input"

```

```

write !,"interesting"
quit

//this procedure can be called from outside the routine
random() public {
  set rand=$RANDOM(10)+1          ; rand is an integer in the range 1-10
  write "Your random number: "_rand
  set name=$$getnumbername(rand)
  write !, "Name of this number: "_name
}

//this procedure can be called from outside the routine
input() public {
  read "Enter a number from 1 to 10: ", input
  set name=$$getnumbername(input)
  write !, "Name of this number: "_name
}

//this procedure can be called only from within this routine
getnumbername(number) {
  set name=$CASE(number,1:"one",2:"two",3:"three",
    4:"four",5:"five",6:"six",7:"seven",8:"eight",
    9:"nine",10:"ten",:"other")
  quit name
}

/* write some interesting values
this procedure can be called from outside the routine
*/
interesting() public {
  write "Today's date: "_$ZDATE($HOROLOG,3)
  write !,"Your installed version: "_$ZVERSION
  write !,"Your username: "_$USERNAME
  write !,"Your security roles: "_$ROLES
}

```

以下の点に注意してください。

- ・ 実際にキャレット (^) で始まる識別子のみがグローバルの名前です。この点については、このページで後述します。ただし、実行されるテキストおよびコード・コメント内では、ルーチン呼び出すときにキャレットを使用するため、一般的に、ルーチンの名前がキャレットで始まっているかのように、ルーチンが参照されます (このページで後述します)。例えば、`demoroutine` というルーチンは、通常、`^demoroutine` と呼び出されます。
- ・ ルーチン名は、そのルーチン内に含める必要はありません。ただし、多くのプログラマは、ルーチンの最初にコメントとして、またはルーチンの最初のラベルとしてルーチン名を含めます。
- ・ ルーチンには、複数のラベル (`random`、`input`、`getnumbername`、および `interesting`) があります。

プロシージャ (この例の場合)、**関数**、および**サブルーチン**の開始点を示すためにラベルを使用します。これらは、特定のコマンドの宛先としても使用できます。

ラベルは、一般的にルーチン内で使用されますが、メソッド内でも使用できます。

ラベルは、エントリ・ポイントまたはタグとも呼ばれます。

- ・ `random` および `input` サブルーチンは、このルーチンに対してプライベートである `getnumbername` サブルーチン呼び出します。

デモンストレーションとして、このルーチンの一部をターミナルで実行できます。最初に、以下はルーチン自体を実行するターミナル・セッションを示しています。

```

TESTNAMESPACE>do ^demoroutine
Use one of the following entry points:
random
input
TESTNAMESPACE>

```

ルーチンを実行するときは、ここに示すようにヘルプ情報が表示されます。ルーチンをこのように記述することは必須ではありませんが、一般的です。ルーチンには、最初のラベルの前に `QUIT` が含まれており、それによって、ユーザがそのルーチン呼び出したときに、そのラベルの前で処理が停止されます。この方法も必須ではありませんが、一般的です。

次に、以下に 1 組みのサブルーチンの動作のしかたを示します。

```
TESTNAMESPACE>do input^demoroutine
Enter a number from 1 to 10: 7
Name of this number: seven
TESTNAMESPACE>do interesting^demoroutine
Today's date: 2018-02-06
Your installed version: IRIS for Windows (x86-64) 2018.1 (Build 513U) Fri Jan 26 2018 18:35:11 EST
Your username: _SYSTEM
Your security roles: %All
TESTNAMESPACE>
```

メソッドには、ルーチンの場合と同じ文、同じラベル、同じコメントを含めることができます。つまり、ルーチンのコンテンツについてここで説明する情報は、メソッドのコンテンツにも当てはまります。

2.3 変数

ObjectScript には、データの保持方法で分類した場合、次の 2 種類の変数があります。

- ローカル変数。データをメモリに保持します。
ローカル変数の範囲は、パブリックまたはプライベートです。
- グローバル変数。データをデータベースに保持します。これらは、グローバルとも呼ばれます。グローバルとのやり取りはすべて、データベースに直接作用します。例えば、グローバルの値を設定した場合、その変更は、格納されているデータに直ちに作用し、値を格納するために別のステップは行われません。同様に、グローバルを削除すると、そのデータはデータベースから直ちに削除されます。

ここでは、説明しませんが特別な種類のグローバルもあります。“[Caché の様々な構文](#)”の“[キャラット \(^\)](#)”を参照してください。

2.3.1 変数名

変数の名前は、以下のルールに従う必要があります。

- 大部分のローカル変数の場合、最初の文字は英字であり、残りの文字は英字または数字です。有効な名前には、`myvar` や `i` があります。
- 大部分のグローバル変数は、最初の文字は常にキャラット (^) です。残りの文字は、英字、数字、またはピリオドです。有効な名前には、`^myvar` や `^my.var` があります。

InterSystems IRIS は、パーセント変数という特別な種類の変数もサポートしています。これは、あまり一般的ではありません。パーセント変数の名前は、パーセント文字 (%) で始まります。パーセント変数は、常にパブリックである点と、プロセス内のすべてのコードから見える点で特別です。これは、呼び出し元スタック内のすべてのメソッドとすべてのプロシージャを含みます。

パーセント変数を定義する場合は、以下のルールに従います。

- ローカル・パーセント変数の場合は、`%z` または `%z` で始まる名前にします。他の名前はシステムで使用するために予約されています。
- グローバル・パーセント変数の場合は、`^%z` または `^%z` で始まる名前にします。他の名前はシステムで使用するために予約されています。

パーセント変数と変数の有効範囲の詳細は、“[変数の可用性と範囲](#)”を参照してください。また、“ObjectScript の使用法”の、“[呼び出し可能なユーザ定義コードモジュール](#)”も参照してください。

名前の詳細およびバリエーションについては、“[識別子のルールとガイドライン](#)” および “ObjectScript の使用法” の “[構文ルール](#)” を参照してください。

2.3.2 変数タイプ

ObjectScript の変数は、弱く動的に型指定されます。変数が動的に型指定されるのは、変数に対して型を宣言する必要がないためであり、変数は有効な値（有効なリテラル値または有効な ObjectScript 式）であればどのような値も取ることができます。変数は、弱く型指定されます。それは、使用法によってそれらの評価方法が決まるためです。

ObjectScript の有効なリテラル値は、次のいずれかの形式です。

- ・ 数値。例：100、17.89、および 1e3
- ・ 引用符で囲まれた文字列。1 組みの引用符 (") で囲まれた一連の文字です。例："my string"
文字列リテラル内に二重引用符を含めるには、"(二重引用符) 文字の前にもう 1 つ " 文字を置きます。例："This string has "quotes" in it."

コンテキストに応じて、文字列が数値として扱われたり、数値が文字列として扱われることがあります。同様に、コンテキストによっては、値がブーリアン (True または False) 値として解釈され、ゼロに評価されるものはすべて False に、それ以外のものはすべて True として処理される場合があります。

クラスを作成する場合、プロパティの型、メソッドの引数などを指定できます。InterSystems IRIS クラスのメカニズムでは、これらの型は、指定したとおりに適用されます。

2.3.3 可変長

変数の値の長さは、3,641,144 文字未満にする必要があります。

2.3.4 変数の存在

変数は、通常、SET コマンドを使用して定義します。前述したように、グローバル変数を定義すると、それはデータベースに直ちに作用します。

グローバル定数は、それを削除した (KILL コマンドを使用して削除する) 場合にのみ未定義になります。これも、即座にデータベースに作用します。

ローカル変数は、以下の 3 つのいずれかの方法で未定義になります。

- ・ 削除 (KILL) された場合。
- ・ それが定義されているプロセスが終了した場合。
- ・ そのプロセス内の範囲から抜けた場合。

変数が定義されているかどうかを判別するには、\$DATA 関数を使用します。例えば、以下は、この関数を使用するターミナル・セッションを示しています。

```
TESTNAMESPACE>write $DATA(x)
0
TESTNAMESPACE>set x=5
TESTNAMESPACE>write $DATA(x)
1
```

最初のステップで、\$DATA を使用して、変数が定義されているかどうかを調べます。0 が表示され、変数が定義されていないことを示します。次に、変数を 5 に設定し、再試行します。今度は関数が 1 を返します。

この例と前の例からわかるように、変数を宣言する必要はありません。必要なのは、SET コマンドのみです。

未定義の変数にアクセスしようとすると、<UNDEFINED> エラーが発生します。以下はその例です。

```
TESTNAMESPACE>WRITE testvar  
  
WRITE testvar  
^  
<UNDEFINED> *testvar
```

2.4 変数の可用性と範囲

ObjectScript は、以下のプログラミング・フローをサポートしており、それは他のプログラミング言語でサポートされているフローと多くの面で類似しています。

1. ユーザがメソッドを呼び出します (通常はユーザ・インタフェースから)。
2. そのメソッドは、いくつかの文を実行し、別のメソッドを呼び出します。
3. そのメソッドは、ローカル変数 A、B、および C を定義します。

変数 A、B、および C は、このメソッド内の範囲内で使用されます。それらは、このメソッドに対してプライベートです。

このメソッドは、グローバル変数 ^D も定義します。

4. 2 番目のメソッドは終了し、制御が最初のメソッドに戻ります。
5. 最初のメソッドが実行を再開します。このメソッドは、変数 A、B、および C を使用できません。それらは定義されていない状態になっています。^D は、直ちにデータベースに保存されたため、使用できます。

上記のプログラム・フローはとても一般的なものです。ただし、InterSystems IRIS では他のオプションも提供されており、それに注意する必要があります。

2.4.1 変数の範囲の概要

変数を定義するメソッドの外でその変数を使用できるかどうかは、いくつかの要因によって決まります。それらを説明する前に、以下の環境の詳細を確認する必要があります。

- ・ InterSystems IRIS インスタンスには、複数のシステム・ネームスペースと、場合によってはユーザが定義した複数のネームスペースなど、複数のネームスペースが含まれています。
ネームスペースとは、その中でコードが実行される環境です。ネームスペースについては、[後で](#)詳細に説明します。
- ・ 1 つのネームスペース内で複数のプロセスを同時に実行できます。一般的なアプリケーションでは、多数のプロセスが同時に実行されています。

以下の表は、変数を使用可能な場所を示しています。

変数の可用性 (種類別)	それが定義されているメソッドの外 (ただし、同じプロセス内)	同じネームスペースの他のプロセス内	同じ InterSystems IRIS インスタンス内の他のネームスペース
ローカル変数、プライベート範囲*	不可	不可	不可
ローカル変数、パブリック範囲	可	不可	不可
ローカル・パーセント変数	あり	不可	不可
グローバル変数 (パーセント以外)	可	可	いいえ (グローバル・マッピングで許可されていない場合)†
グローバル・パーセント変数	可	可	可

*既定では、メソッド内で定義された変数は、前述したとおり、そのメソッドに対してプライベートです。また、メソッドでは、変数をパブリック変数として宣言できます。ただし、この方法はお勧めできません。“[PublicList](#)”を参照してください。

†各ネームスペースには、特定の目的のための既定のデータベースがあり、追加のデータベースへのアクセスを提供するマッピングを設定できます。その結果、グローバル変数がグローバル・パーセント変数でない場合でも、それを複数のネームスペースで使用可能にすることができます。“[ネームスペースとデータベース](#)”を参照してください。

2.5 多次元配列

ObjectScript では、どの変数も InterSystems IRIS 多次元配列 (配列とも呼ぶ) にすることができます。多次元配列は、通常、何らかの関連を持つ一連の値を保持することを目的としています。ObjectScript には、値への便利で高速なアクセスを可能にする[コマンド](#)と[関数](#)があります。

多次元配列を直接操作するかどうかは、使用するシステム・クラスと好みによって異なります。InterSystems IRIS には、一連の関連する値のためのコンテナが必要な場合に使用するクラスベースの代替手段があります。“[コレクション・クラス](#)”を参照してください。

2.5.1 基本

多次元配列は、添え字で定義される、任意の数のノードで構成されます。以下の例では、配列のいくつかのノードが設定され、その配列のコンテンツが印刷されます。

ObjectScript

```
set myarray(1)="value A"
set myarray(2)="value B"
set myarray(3)="value C"
zwrite myarray
```

この例は、一般的な例を示しています。メモ：

- この配列には 1 つの添え字があります。この場合、添え字は整数 1、2、および 3 です。
- 事前に配列の構造を宣言する必要はありません。
- myarray は、その配列自体の名前です。

- ・ ObjectScript には、配列全体または特定ノードに対して作用できるコマンドおよび関数があります。以下はその例です。

ObjectScript

```
kill myarray
```

また、特定のノードおよびその子のノードを削除することもできます。

- ・ 以下のバリエーションは、`^myglobal` という名前のグローバル配列のいくつかの添え字を設定します。つまり、これらの値がディスクに書き込まれます。

ObjectScript

```
set ^myglobal(1)="value A"  
set ^myglobal(2)="value B"  
set ^myglobal(3)="value C"
```

- ・ グローバル参照には長さの制限があります。この制限は、グローバル名の長さ、および添え字の長さや数に影響します。制限を超えた場合、`<SUBSCRIPT>` エラーが発生します。“グローバルの使用法”の“[グローバル参照の最大長](#)”を参照してください。
- ・ ノードの値の長さは、3,641,144 文字未満にする必要があります。

多次元配列は、定義済みノードごとに 1 つの予約済みメモリ位置を持っており、それ以外は持っていません。グローバルの場合、それが使用するディスク領域はすべて動的に割り当てられます。

2.5.2 構造のバリエーション

上記の例は、配列の一般的な形を示しています。以下の使用可能なバリエーションに注意してください。

- ・ 添え字の数に制限はありません。以下はその例です。

ObjectScript

```
Set myarray(1,1,1)="grandchild of value A"
```

- ・ 添え字は文字列にすることができます。以下も有効です。

ObjectScript

```
set myarray("notes to self","2 Dec 2010")="hello world"
```

2.5.3 使用上の注意

ObjectScript の学習者がよく犯す誤りは、グローバルと配列を混同することです。どの変数もローカルかグローバルのいずれかであり、さらに添え字ありか添え字なしのいずれかであることを忘れないでください。以下の表は、その例を示しています。

変数の種類	例と注意
ローカル変数 (添え字なし)	Set MyVar=10 このような変数はきわめて一般的です。変数の大部分はこのような変数です。
ローカル変数 (添え字付き)	Set MyVar(1)="alpha" Set MyVar(2)="beta" Set MyVar(3)="gamma" このようなローカル配列は、一連の関連するデータを渡す場合に便利です。
グローバル変数 (添え字なし)	Set ^MyVar="saved note" 実際、グローバルには通常添え字があります。
グローバル変数 (添え字付き)	Set ^MyVar(\$USERNAME,"Preference 1")=42

2.6 値または参照による変数渡し

メソッドを呼び出す場合、値または参照によって、そのメソッドに変数の値を渡すことができます。多くの場合、これらの変数は添え字なしのローカル変数であるため、このセクションでは最初にそれらについて説明します。

他のプログラミング言語の場合と同様に、InterSystems IRIS は、各ローカル変数の値を格納するメモリ位置を持っています。変数の名前は、メモリ位置のアドレスとして機能します。

添え字なしのローカル変数をメソッドに渡すときは、その変数を値によって渡します。つまり、システムによってその値のコピーが作成され、元の値が変更されることはありません。代わりに、メモリ・アドレスを渡すには、引数リストの変数の名前の直前にピリオドを配置します。

この例を示すため、Test.Parameters というクラス内の次のメソッドを見てみましょう。

Class Member

```
ClassMethod Square(input As %Integer) As %Integer
{
    set answer=input*input
    set input=input + 10
    return answer
}
```

変数を定義し、それをこのメソッドに値によって渡すとします。

```
TESTNAMESPACE>set myVariable = 5
TESTNAMESPACE>write ##class(Test.Parameters).Square(myVariable)
25
TESTNAMESPACE>write myVariable
5
```

反対に、変数を参照によって渡すとします。

```
TESTNAMESPACE>set myVariable = 5
TESTNAMESPACE>write ##class(Test.Parameters).Square(.myVariable)
25
TESTNAMESPACE>write myVariable
15
```

以下のメソッドを考えてみます。これは、受け取った引数の内容を書き込みます。

```
ClassMethod WriteContents(input As %String)
{
    zwrite input
}
```

ここで、3 つのノードが含まれる配列があるとします。

```
TESTNAMESPACE>zwrite myArray
myArray="Hello"
myArray(1)="My"
myArray(2)="Friend"
```

この配列をメソッドに値によって渡すと、最上位のノードのみが渡されます。

```
TESTNAMESPACE>do ##class(Test.Parameters).WriteContents(myArray)
input="Hello"
```

この配列をメソッドに参照によって渡すと、配列全体が渡されます。

```
TESTNAMESPACE>do ##class(Test.Parameters).WriteContents(.myArray)
input="Hello"
input(1)="My"
input(2)="Friend"
```

グローバルの 1 つのノードの値をメソッドに渡すことができます。

```
TESTNAMESPACE>zwrite ^myGlobal
^myGlobal="Start"
^myGlobal(1)="Your"
^myGlobal(2)="Engines"
TESTNAMESPACE>do ##class(Test.Parameters).WriteContents(^myGlobal)
input="Start"
```

グローバルをメソッドに参照によって渡そうとすると、構文エラーが発生します。

```
TESTNAMESPACE>do ##class(Test.Parameters).WriteContents(^myGlobal)
^
<SYNTAX>
```

以下の表は、そのすべてのバリエーションを示しています。

変数の種類	値渡し	参照渡し
ローカル変数 (添え字なし)	これらの変数を渡す標準的な方法	可能
ローカル (添え字付き) (配列)	1 つのノードの値を渡す	これらの変数を渡す標準的な方法
グローバル変数 (添え字付き、または添え字なし)	1 つのノードの値を渡す	この方法では渡せません (グローバルのデータがメモリ内にありません)

2.7 演算子

このセクションでは、ObjectScript の演算子の概要について説明します。[よく知られているもの](#)と、[あまり知られていないもの](#)があります。

ObjectScript で演算子の評価順序は、必ず左から右です。したがって、式の演算は表示された順番で実行されます。式で明示的に小括弧を使用して、特定の演算子を先に処理させることができます。

通常、括弧は、必ずしも必要でない場所にも使用します。これは、他のプログラマに (および後で作成者自身に)、コードの意図を明確にすることができるので便利です。

2.7.1 よく知られている演算子

ObjectScript には、一般的なアクティビティのための以下の演算子があります。

- ・ 算術演算子：加算 (+)、減算 (-)、除算 (/)、乗算 (*)、整数除算 (\)、剰余 (#)、および指数 (**)
- ・ 単項演算子：正 (+)、および負 (-)
- ・ 文字列連結演算子 (⋈)
- ・ 論理比較演算子：等しい (=)、より大きい (>)、以上 (>=)、より小さい (<)、以下 (<=)
- ・ 論理補数演算子 (!)

これは、論理値の直前および論理比較演算子の直前に使用できます。

- ・ 論理値を組み合わせる演算子：AND (&&)、OR (||)

ObjectScript では、これらの演算子それぞれの古くて効率的でない形式 (&& 演算子の &、|| 演算子の !) もサポートされています。これらの古い形式は、既存のコードに使用されていることがあります。

2.7.2 あまり知られていない演算子

ObjectScript には、いくつかの言語に同等のものがない演算子もあります。最も重要なものは以下のとおりです。

- ・ パターン・マッチング演算子 (?) は、左オペランドの文字パターンが、その右オペランドのパターンを使用しているかどうかを判断します。パターンが発生する回数、代替パターン、パターンの入れ子などを指定できます。

例えば、以下は、文字列 (testthis) が米国の社会保障番号の書式である場合は、値 1 (True) を、それ以外の場合は 0 を返します。

ObjectScript

```
Set testthis="333-99-0000"
Write testthis ?3N1 "-" 2N1 "-" 4N
```

これは、入力データの妥当性を確保するための価値のあるツールであり、クラス・プロパティの定義内で使用できます。

- ・ 二項包含関係演算子 (I) は、右のオペランドの一連の文字が、左の文字の部分文字列であるかどうかによって、1 (True) または 0 (False) を返します。以下はその例です。

ObjectScript

```
Set L="Steam Locomotive",S="Steam"
Write L[I S
```

- ・ 二項後続関係演算子 (J) は、左のオペランドの文字が、ASCII 文字順で右のオペランドの文字の後に来るかどうかを判断します。
- ・ 二項前後関係演算子 (J J) は、左のオペランドが数値添え字の照合順序で右のオペランドの後に順番に並んでいるかを判定します。
- ・ 間接演算子 (@) により、コマンド引数、変数名、添え字リスト、またはパターンの一部またはすべてを実行時に動的に置き換えることができます。InterSystems IRIS は、関連するコマンドを実行する前に、置換を実行します。

2.8 コマンド

このセクションでは、これから使用する可能性が高いコマンド、および ObjectScript でよく使用されているコマンドの概要について説明します。これらには、他の言語に類似したコマンドと、他の言語には同等のものがないコマンドがあります。

コマンドの名前では大文字小文字は区別されませんが、規約によって実行されるテキストではすべて大文字で表記します。

2.8.1 よく知られているコマンド

ObjectScript には、以下のようなよく知られたタスクを実行するためのコマンドがあります。

- ・ 変数を定義するには、前述のように SET を使用します。
- ・ 変数を削除するには、前述のように KILL を使用します。
- ・ ロジックのフローを制御するには、以下のコマンドを使用します。
 - IF、ELSEIF、および ELSE。これらは連携して機能します。
 - FOR
 - WHILE。単独で使用できます。
 - DO および WHILE。これらは一緒に使用できます。
 - QUIT。これも値を返すことができます。

フローを制御するコマンドはほかにもありますが、それらはあまり使用されません。

- ・ エラーをトラップするには、TRY および CATCH を使用します。これらは連携して機能します。
- ・ 値を書き込むには、WRITE を使用します。これは、現在のデバイス（例えば、ターミナルやファイル）に値を書き込みます。

このコマンドを引数なしで使用した場合、すべてのローカル変数の値を書き込みます。これは、特にターミナルで便利です。

このコマンドは、出力を配置する一連の形式制御コード文字を使用できます。既存のコードでは、新しい行を開始するために感嘆符が使用されている場合があります。以下はその例です。

ObjectScript

```
write "hello world",!,"another line"
```

- ・ 現在のデバイス（例えば、ターミナル）から値を読み取るには、READ を使用します。
- ・ 主デバイス以外のデバイスを使用するには、以下のコマンドを使用してください。
 - OPEN は、デバイスを使用可能にします。
 - USE は、使用可能になったデバイスを WRITE および READ で使用するために現在のデバイスとして指定します。
 - CLOSE は、デバイスを使用不可にします。
- ・ 同時処理を制御するには、LOCK を使用します。InterSystems IRIS のロック管理システムは、他の言語の同様のシステムとは異なる点に注意してください。この動作のしくみを再確認することが重要になります。[“ロックと並行処理の制御”](#)を参照してください。

このコマンドは、複数のプロセスが同一の変数などの項目にアクセスする可能性がある場合に使用します。

- ・ トランザクションを管理するには、TSTART、TCOMMIT、TROLLBACK、および関連コマンドを使用します。
- ・ デバッグするには、ZBREAK および関連コマンドを使用します。
- ・ 実行を中断するには、HANG を使用します。

2.8.2 多次元配列で使用するコマンド

ObjectScript では、以下の方法で多次元配列を操作できます。

- ・ ノードを定義するには、SET コマンドを使用します。
- ・ 個別のノードまたはすべてのノードを削除するには、KILL コマンドを使用します。

例えば、以下は、多次元配列全体を削除します。

ObjectScript

```
kill myarray
```

以下は、対照的にノード myarray("2 Dec 2010") およびその子すべてを削除します。

ObjectScript

```
kill myarray("2 Dec 2010")
```

- ・ グローバルまたはグローバル・ノードを削除するが、その下位サブノードは削除しない場合は、ZKILL を使用します。
- ・ 多次元配列のすべてのノードに繰り返し処理を行い、それらすべてを書き込むには、ZWRITE を使用します。これは、特にターミナルで便利です。以下のターミナル・セッションの例は、出力がどのようなものになるのかを示しています。

```
TESTNAMESPACE>ZWRITE ^myarray
^myarray(1)="value A"
^myarray(2)="value B"
^myarray(3)="value C"
```

この例では、ローカル変数ではなくグローバル変数を使用しますが、どちらも多次元配列にすることができることを忘れないでください。

- ・ 一連のノードを1つの多次元配列から別の多次元配列に、ターゲットに既存のノードを残したままコピーするには、MERGE を使用します。例えば、以下のコマンドはメモリ内の配列 (sourcearray) 全体を新しいグローバル (^mytestglobal) にコピーします。

ObjectScript

```
MERGE ^mytestglobal=sourcearray
```

これは、コードをデバッグしているときに、使用している配列のコンテンツを調べるのに便利です。

2.9 特殊変数

このセクションでは、InterSystems IRIS のいくつかの特殊変数について説明します。これらの変数の名前では、大文字と小文字は区別されません。

いくつかの特殊変数は、コードが実行されている環境に関する情報を提供します。これは、以下の項目が含まれます。

- ・ \$HOROLOG。これには、オペレーティング・システムから提供される、現在のプロセスの日付と時刻が格納されます。“日付と時刻の値”を参照してください。
- ・ \$USERNAME および \$ROLES。これには、現在使用しているユーザ名と、そのユーザが属しているロールに関する情報が格納されます。

ObjectScript

```
write "You are logged in as: ", $USERNAME, !, "And you belong to these roles: ", $ROLES
```

- ・ \$VERSION。これには、現在動作している InterSystems IRIS のバージョンを表す文字列が格納されます。

そのほかに \$JOB、\$ZTIMEZONE、\$IO、および \$ZDEVICE があります。

他の変数は、コードの処理状態に関する情報を提供します。それらには、\$STACK、\$TLEVEL、\$NAMESPACE、および \$ZERROR があります。

2.9.1 \$SYSTEM 特殊変数

特殊変数 \$SYSTEM は、一連の多数のユーティリティ・メソッドへの簡単なアクセスを提供します。

\$SYSTEM 特殊変数は、%SYSTEM パッケージのエイリアスであり、そのパッケージには、幅広いニーズに対応するクラス・メソッドを提供するクラスが含まれています。%SYSTEM のメソッドを参照する一般的な方法は、\$SYSTEM 変数を使用する参照を構築することです。例えば、以下のコマンドは %SYSTEM.OBJ クラスの SetFlags() メソッドを実行します。

ObjectScript

```
DO $SYSTEM.OBJ.SetFlags("ck")
```

特殊変数の名前は、クラス名やそのメンバ名とは異なり、大文字と小文字が区別されないため、以下のコマンドはすべて等価です。

ObjectScript

```
DO ##class(%SYSTEM.OBJ).SetFlags("ck")
DO $System.OBJ.SetFlags("ck")
DO $SYSTEM.OBJ.SetFlags("ck")
DO $system.OBJ.SetFlags("ck")
```

このクラスにはすべて、Help() メソッドがあり、これはそのクラスで使用可能なメソッドのリストを出力できます。以下はその例です。

```
TESTNAMESPACE>d $system.OBJ.Help()
'Do $system.OBJ.Help(method)' will display a full description of an individual method.

Methods of the class: %SYSTEM.OBJ

CloseObjects()
  Deprecated function, to close objects let them go out of scope.

Compile(classes,qspec,&errorlog,recurse)
  Compile a class.

CompileAll(qspec,&errorlog)
  Compile all classes within this namespace
....
```

また、メソッドの名前を Help() の引数として使用することもできます。以下はその例です。

```
TESTNAMESPACE>d $system.OBJ.Help("Compile")
Description of the method: class Compile: %SYSTEM.OBJ

Compile(classes:%String="", qspec:%String="", &errorlog:%String, recurse:%Boolean=0)
Compile a class.
<p>Compiles the class <var>classes</var>, which can be a single class, a comma separated list,
a subscripted array of class names, or include wild cards. If <var>recurse</var> is true then
do not output the initial 'compiling' message or the compile report as this is being called inside
another compile loop.<br>
<var>qspec</var> is a list of flags or qualifiers which can be displayed with
'Do $system.OBJ.ShowQualifiers()'
and 'Do $system.OBJ.ShowFlags()'
```

2.10 ロックと並行処理の制御

マルチプロセス・システムでの重要な機能の 1 つに、並行処理の制御があります。これは、異なるプロセスがデータの特定の要素を同時に変更し、破損に至ることを防止する機能です。そのために、ObjectScript にはロック管理システムが用意されています。このセクションでは、簡単な概要を説明します。

“[ロック、グローバル、およびネームスペース](#)” も参照してください。

2.10.1 基本

基本のロック・メカニズムは、LOCK コマンドです。このコマンドの目的は、あるプロセス内のアクティビティを、別のプロセスが処理継続 OK の合図を送るまで遅延することです。

ロックそれ自体では他のプロセスによる関連データ変更を防止しないことを理解しておくことが重要です。つまり、InterSystems IRIS は一方的なロックを強制実施しないということです。規約では、ロックが機能するには、互いに競合するプロセスがすべて同じロック名でロックを実装していることが要求されています。

LOCK コマンドは、ロックの作成（プロセスが所有していた以前のすべてのロックの置換）、ロックの追加、特定のロックの削除、およびプロセスが所有するすべてのロックの削除に使用できます。

ここでの簡単な説明のために、LOCK コマンドに次の引数を使用します。

- ・ ロック名。ロック名は、任意の名前ですが、共通の規則として、プログラマはロックする項目の名前と同じロック名を使用します。通常、ロックする項目は、グローバルまたはグローバルのノードになります。
- ・ オプションのロック・タイプ（既定のタイプ以外のロックを作成する場合）。複数のロック・タイプがあり、それぞれの動作が異なります。
- ・ オプションのタイムアウト引数。この引数では、ロック操作の試行をタイムアウトにするまでの待機時間を指定します。既定では、InterSystems IRIS は無限に待機します。

次に一般的なロック・シナリオについて説明します。プロセス A が LOCK コマンドを発行し、InterSystems IRIS はロックの作成を試行します。プロセス B が指定されたロック名のロックを既に所有している場合、プロセス A は一時停止します。具体的には、プロセス A の LOCK コマンドは復帰しなくなり、コードの後続行が実行できなくなります。プロセス B がロックを解放すると、プロセス A の LOCK コマンドが復帰して実行を継続できるようになります。

システムは、永続オブジェクト（このドキュメントで後述）を操作するときや、特定の InterSystems SQL コマンドを使用するときなど、多くの場合に自動的に LOCK コマンドを使用します。

2.10.2 ロック・テーブル

InterSystems IRIS は、現在のすべてのロックとロック所有プロセスを記録するために、システム規模のテーブルをメモリ内に保持しています。このテーブル（ロック・テーブル）は、管理ポータルからアクセスできます。管理ポータルでは、ロッ

クが表示と、まれに必要なロックの削除を実行できます。特定のプロセスが、ロック名が異なる複数のロックを所有している可能性がある点に注意してください（複数のロックが同じ名前の場合もあります）。

プロセスが終了すると、システムは、そのプロセスが所有するすべてのロックを自動的に解放します。そのため、通常は、管理ポータルからロックを削除する必要はありません。ただし、アプリケーション・エラーが発生した場合を除きます。

ロック・テーブルは、固定サイズを超過することはありません（このサイズは指定できます）。詳細は、“監視ガイド”の“[ロックの監視](#)”を参照してください。そのため、ロック・テーブルが一杯になり、それ以上のロックが不可能になることがあります。この場合、InterSystems IRIS は、**messages.log** ファイルに以下のエラー・メッセージを書き込みます。

```
LOCK TABLE FULL
```

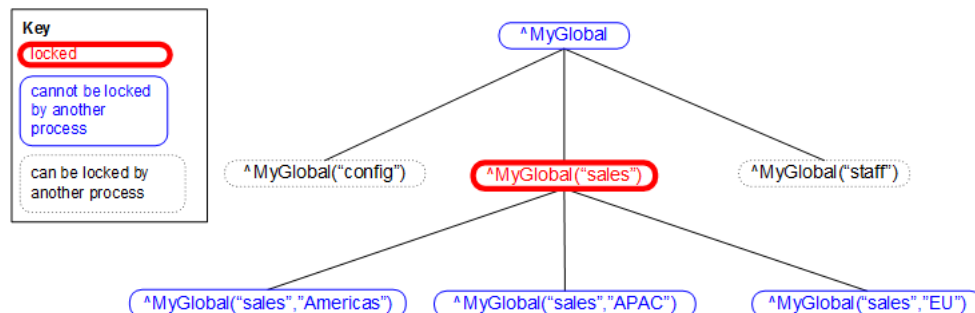
通常は、ロック・テーブルが一杯になっても、アプリケーション・エラーとは見なされません。InterSystems IRIS には、ロック・キューも用意されているため、プロセスはロック・テーブルにロックを追加する領域ができるまで待機します。

ただし、2つのプロセスが、既に別のプロセスでロックされている変数に対して増分ロックをアサートすると、デッドロックと呼ばれる状態になり、その状態はアプリケーション・プログラミング・エラーと見なされます。詳細は、“[ObjectScript の使用法](#)”の“[デッドロックの回避](#)”を参照してください。

2.10.3 ロックと配列

配列をロックする場合は、配列全体をロックすることも、配列内の1つ以上のノードをロックすることもできます。配列ノードをロックすると、他のプロセスは、そのノードに従属するどのノードもロックできなくなります。また、他のプロセスは、ロックされたノードの直接の祖先もロックできなくなります。

次の図に例を示します。



2.10.4 ロック・タイプの概要

ロックの作成時には、ロックの特性を制御する、ロック・タイプ・コードの組み合わせを指定できます。このセクションでは、ロック・タイプの重要なコンセプトの一部について説明します。

ロック・タイプによっては、同じロック名で複数のロックを作成することも可能です。これらのロックは同一プロセスが所有することも、別々のプロセスが所有することもできます（繰り返しになりますが、これは、ロック・タイプによって異なります）。ロック・テーブルには、これらすべてについての情報が表示されます。

あらゆるロックは、排他（既定）または共有のどちらかになります。これらのタイプは、次に示す重要な意味を持ちます。

- あるプロセスが特定のロック名の付いた排他ロックを所有しているときに、その他のプロセスは、そのロック名の付いたロックを取得できません。
- あるプロセスが特定のロック名の付いた共有ロックを所有しているときに、その他のプロセスは、そのロック名の付いた共有ロックを取得できます。ただし、他のプロセスはそのロック名で排他的なロックを取得できません。

一般に、排他ロックは、値を変更する予定があり、他のプロセスはその値の読み取りまたは変更を試行してはいけないことを示すために使用されます。共有ロックの目的は、値を読み取る予定があることを示し、その他のプロセスがその値の変更を試行してはいけないことを示すことです。ただし、その他のプロセスがその値を読み取ることは可能です。

さらに、あらゆるロックは、非エスカレート(既定)またはエスカレートのどちらかになります。エスカレート・ロックの目的は、メモリを消費し、**ロック・テーブル**が埋め尽くされる可能性が大きくなる大量のロックの管理を簡単にすることです。エスカレート・ロックは、同じ配列に含まれる複数のノードをロックする場合に使用します。エスカレート・ロックの場合、あるプロセスがある配列の兄弟ノードに特定の数(既定では 1,000)を超えるロックを作成すると、InterSystems IRIS は、個別のロック名をすべて削除して、それらを親レベルで新しい 1 つのロックに置き換えます。例えば、`^MyGlobal("sales","EU",salesdate)` という形式で `salesdate` が日付を表しているロックが 1,000 個あるとします。同じプロセスが、この形式で別のロックを作成しようとする(すべてのロックがエスカレート・ロックの場合)、InterSystems IRIS は、これらのロックをすべて削除してから、`^MyGlobal("sales","EU")` という名前のロックに置き換えます。ロック・テーブルは、この新しいロック用のロック・カウントを維持します。このロック・カウントは、現在 1,001 ですが、同じ形式の別のロック名を追加すると、ロック・テーブルでは、ロック名 `^MyGlobal("sales","EU")` のロック・カウントがインクリメントされます。同様に、同じ形式のロック名を削除すると、ロック・テーブルではこのロック・カウントがデクリメントされます。

InterSystems IRIS には、トランザクションの範囲内で特別な方法で扱われるロックの追加のサブタイプがあります。これらのロックの詳細と、全般的なロックの説明は、“ObjectScript リファレンス”の“**LOCK**”を参照してください。ロックしきい値(既定値は 1000)の指定の詳細は、“構成パラメータ・ファイル・リファレンス”の“**LockThreshold**”を参照してください。

2.11 システム関数

このセクションでは、ObjectScript で最もよく使用されるシステム関数のいくつかに焦点を当てます。

これらの関数の名前では、大文字と小文字は区別されません。

InterSystems IRIS クラス・ライブラリでは、関数を使用するのと同じように使用できる多数のユーティリティ・メソッドも提供されています。特定の用途のメソッドを見つけるには、“インターシステムズ・プログラミング・ツールの索引”を使用してください。

“**日付と時刻の値**”も参照してください。

2.11.1 値の選択

以下の関数を使用すると、入力に対して 1 つの値を選択できます。

- ・ `$CASE` は、指定されたテスト評価式を、一連の比較値と比較し、一致した比較値と関連付けられた返り値を返します。以下はその例です。

```
TESTNAMESPACE>set myvar=1

TESTNAMESPACE>write $CASE(myvar,0:"zero",1:"one",:"other")
one
```

- ・ `$SELECT` は、一連の式を評価し、最初の True 式に関連付けられた返り値を返します。以下はその例です。

```
TESTNAMESPACE>set myvar=1

TESTNAMESPACE>write $SELECT(myvar=0:"branch A",1=1:"branch B")
branch B
```

2.11.2 存在の関数

以下の関数を使用して、変数の存在または変数のノードの存在をテストできます。

- ・ 特定の変数が存在するかどうかを判別するには、\$DATA 関数を使用します。
複数のノードを格納する変数に対しては、この関数は、指定されたノードが存在するかどうか、および指定されたノードに値と子ノードがあるかどうかを示すことができます。
- ・ 変数の値を取得する（変数が存在する場合）または既定値を取得する（存在しない場合）には、\$GET 関数を使用します。

2.11.3 リスト関数

ObjectScript には、ネイティブのリスト形式が用意されています。以下の関数を使用して、これらのリストを作成および操作できます。

- ・ \$LISTBUILD は、リストと呼ばれる特別な種類の文字列を返します。これは、他の種類のリスト（コンマ区切りリストなど）と区別するために \$LIST 形式と呼ばれることもあります。
\$LIST リストの操作には、ObjectScript のリスト関数の使用のみがサポートされます。この種のリストの内部構造は文書化されていません。また、予告なしに変更される場合があります。
- ・ \$LIST は、リスト要素を返します。また、リスト要素を置換するために使用することもできます。
- ・ \$LISTLENGTH は、リスト内の要素の数を返します。
- ・ \$LISTFIND は、指定されたリスト内の指定された要素の位置を返します。

リスト関数は、ほかにもあります。

リストにない値を指定してリスト関数を使用すると、<LIST> エラーを受け取ります。

注釈 システム・クラス %Library.List は、\$LISTBUILD によって返されるリストと同じです。つまり、クラスに %Library.List タイプのプロパティがある場合、ここに示す関数を使用してそのプロパティを操作します。このクラスは、その短い名前である %List によって参照できます。

InterSystems IRIS には、\$LISTBUILD によって返されるリストとは異なる他のリスト・クラスもあります。これらは、クラスを操作する場合に便利です。概要は、“[コレクション・クラス](#)”を参照してください。

2.11.4 文字列関数 (1)

ObjectScript には、文字列を効率的に使用するための一連の広範な関数も用意されています。

- ・ \$EXTRACT は、文字カウントを使用して、部分文字列を返すか、置換します。
- ・ \$FIND は、値により部分文字列を検索し、文字列の最終位置を指定する整数を返します。
- ・ \$JUSTIFY は、左にスペースを埋め、右揃えした文字列を返します。
- ・ \$ZCONVERT は、文字列の形式を変換します。大文字と小文字の変換（大文字変換、小文字変換、タイトル文字変換など）とコード変換（さまざまな形式でコード化された文字間の変換）をサポートします。
- ・ \$TRANSLATE は、文字単位の置換を実行し、指定した文字列を変更します。
- ・ \$REPLACE は、1 つの文字列内で文字列単位の置換を実行し、新しい文字列を返します。
- ・ \$PIECE は、文字で区切られた文字列（分割化された文字列 と呼ぶ）から、部分文字列を返します。以下は、部分文字列の抽出方法の例を示しています。

ObjectScript

```
SET mystring="value 1^value 2^value 3"
WRITE $PIECE(mystring,"^",1)
```

- ・ \$LENGTH は使用されるパラメータによって、指定した文字列内の文字数あるいは指定した文字列内の区切られた部分文字列の数を返します。

以下はその例です。

ObjectScript

```
SET mystring="value 1^value 2^value 3"
WRITE !, "Number of characters in this string: "
WRITE $LENGTH(mystring)
WRITE !, "Number of pieces in this string: "
WRITE $LENGTH(mystring,"^")
```

2.11.5 多次元配列の操作

以下の関数を使用して多次元配列全体を操作できます。

- ・ \$ORDER を使用すると、多次元配列内の各ノードに順番にアクセスできます。
- ・ \$QUERY を使用すると、サブノード間を移動して、配列内の各ノードと各サブノードにアクセスできます。

配列内の個別ノードを操作するには、前述した、どの関数も使用できます。特に次のことが可能です。

- ・ \$DATA は、指定されたノードが存在するかどうか、および指定されたノードに子ノードがあるかどうかを示すことができます。
- ・ \$GET は、指定されたノードの値を取得するか、それ以外の場合は既定値を取得します。

2.11.6 文字値

文字列を作成するときに、タイプできない文字を含めることが必要な場合があります。そのような場合は、\$CHAR を使用します。

整数を指定すると、\$CHAR は、対応する ASCII または Unicode 文字を返します。一般的な使用法は以下のとおりです。

- ・ \$CHAR(9) はタブです。
- ・ \$CHAR(10) は改行です。
- ・ \$CHAR(13) はキャリッジ・リターンです。
- ・ \$CHAR(13,10) はキャリッジ・リターンと改行のペアです。

関数 \$ASCII は、指定された文字の ASCII 値を返します。

2.12 日付と時刻の値

このセクションでは、ObjectScript の日付と時刻の値の簡単な概要について説明します。

2.12.1 ローカル時刻

現在のプロセスの日付と時刻にアクセスするには、\$HOROLOGY 特殊変数を使用します。これが原因となって、多くの InterSystems IRIS アプリケーションでは、日付と時刻はこの変数で使われる形式で格納され、送信されます。この形式は、通常、\$H 形式または \$HOROLOGY 形式と呼ばれます。

\$HOROLOGY は、オペレーティング・システムから日付と時刻を取得するため、常にローカル・タイムゾーンになります。

InterSystems IRIS クラス・ライブラリには、ODBC など一般的な形式で日付を表すデータ型クラスが含まれており、多くのアプリケーションは \$H 形式の代わりにそれらを使用します。%Library.PosixTime データ型クラスによって POSIX 時刻がサポートされています。新しいアプリケーションでは、POSIX 時刻を使用して日/時の値を表すようにしてください。

2.12.2 UTC 時刻

InterSystems IRIS には、\$ZTIMESTAMP 特殊変数も用意されています。これには、現在の日付と時刻が協定世界時 (UTC) 値で、\$H 形式で格納されます。UTC は、時刻と日付の世界標準です。この値は、ローカル時刻 (および日付) の値と異なる場合がほとんどです。

2.12.3 日付・時刻の変換

ObjectScript には、日付と時刻の値を変換するための関数もあります。

- 関数 \$ZDATE は、\$H 形式で日付を指定された場合、指定した形式で日付を表す文字列を返します。

以下はその例です。

```
TESTNAMESPACE>WRITE $ZDATE($HOROLOG,3)
2010-12-03
```

- 関数 \$ZDATETIME は、\$H 形式で日付と時刻を指定された場合、指定した形式で日付と時刻を表す文字列を返します。

以下はその例です。

```
TESTNAMESPACE>WRITE $ZDATETIME($HOROLOG,3)
2010-12-03 14:55:48
```

- 他の形式の日付と時刻の文字列を指定された場合、関数 \$ZDATEH および \$ZDATETIMEH は、それらを \$H 形式に変換します。
- 関数 \$ZTIME は時刻を \$H 形式から変換し、\$ZTIMEH は時刻を \$H 形式に変換します。

2.12.4 \$H 形式の詳細

\$H 形式は、コンマで区切られた数値のペアです。例：54321,12345

- 最初の数値は、1840 年 12 月 31 日からの日数です。つまり、日付の番号 1 は 1841 年 1 月 1 日です。この数値は常に整数です。
- 2 番目の数値は、指定された日付の午前 0 時からの秒数です。
\$NOW() などのいくつかの関数では、小数部分が提供されます。

開始日の説明を含む詳細は、“ObjectScript リファレンス”の“\$HOROLOG”を参照してください。

2.13 マクロとインクルード・ファイルの使用

前述のように、マクロを定義し、後でそれらのマクロをその同じクラスまたはルーチン内で使用できます。一般的には、それらはインクルード・ファイルで定義します。

2.13.1 マクロ

ObjectScript では、置換を定義するマクロがサポートされます。その定義は、値、あるコード行の全体、(##continue 指示文を含む) 複数の行のいずれかです。

マクロを定義するには、#define 指示文または他のプリプロセッサ指示文を使用します。以下はその例です。

ObjectScript

```
#define macroname <definition>
```

マクロを参照するには、以下の構文を使用します。

```
$$$macroname
```

または以下のようにします。

```
$$$macroname(arguments)
```

マクロは、整合性を確保するために使用します。以下はその例です。

ObjectScript

```
#define StringMacro "Hello, World!"
write $$$StringMacro
```

マクロで何を行えるのかを示すために、内部で使用されるマクロの定義の例を以下に示します。

```
#define CALL(%C,%A) $$$INTCALL(%C,%A,Quit sc)
```

このマクロは、多くのマクロと同様に、引数を受け入れます。また、別のマクロを参照します。

システム・クラスによっては、マクロを広範に使用するものがあります。

プリプロセッサ指示文の詳細は、“ObjectScript の使用法” の “[ObjectScript マクロとマクロ・プリプロセッサ](#)” を参照してください。

注釈 管理ポータルに、ルーチンを含むインクルード・ファイルのリストが表示されます。ただし、インクルード・ファイルは実行ファイルではないため、実際にはルーチンではありません。

2.13.2 インクルード・ファイル

クラスまたはルーチン内でマクロを定義し、後でそれらのマクロをその同じクラスまたはルーチン内で使用できます。より一般的には、それらは中央の場所で定義します。このためには、インクルード・ファイルを作成して使用します。インクルード・ファイルは、マクロを定義し、他のインクルード・ファイルをインクルードできます。インクルード・ファイルは、拡張子 .inc が付いたドキュメントです。

インクルード・ファイルを作成したら、以下のことを実行できます。

- ・ ルーチンの最初でインクルード・ファイルをインクルードします。そのルーチンは、インクルード・ファイル内で定義されているマクロを参照できます。
- ・ クラスの最初でインクルード・ファイルをインクルードします。そのクラスのメソッドは、そのマクロを参照できます。
- ・ メソッドの最初でインクルード・ファイルをインクルードします。そのメソッドは、そのマクロを参照できます。

以下に示すのは、システム・インクルード・ファイルの一部です。

ObjectScript

```
/// Create a success %Status code
#define OK 1

/// Return true if the %Status code is success, and false otherwise
/// %sc - %Status code
#define ISOK(%sc) (+%sc)

/// Return true if the %Status code if an error, and false otherwise
/// %sc - %Status code
#define ISERR(%sc) ('%sc)
```

インクルード・ファイルをルーチンまたはメソッドにインクルードするには、`#include` 指示文を使用します。以下に例を示します。

ObjectScript

```
#include myincludefile
```

クラスの定義の開始部でインクルード・ファイルをインクルードする場合、指示文にシャープ記号を含めません。以下に例を示します。

```
Include myincludefile
```

または

```
Include (myincludefile, yourincludefile)
```

2.14 ルーチンの使用

ルーチンは ObjectScript プログラムと見なすことができます。ルーチンを一から記述することも、クラスのコンパイル時にルーチンを自動生成することもできます。

2.14.1 プロシージャ、関数、およびサブルーチン

ObjectScript ルーチン内では、ラベルは、以下のコード・ユニットの 1 つの開始ポイントを定義します。

- ・ プロシージャ (オプションで値を返します)。プロシージャで定義される変数は、そのプロシージャに対してプライベートです。これは他のコードから使用できないことを意味します。これは、関数およびサブルーチンの場合は異なります。

プロシージャは、プロシージャ・ブロックと呼ばれます。

- ・ 関数 (値を返します)。
- ・ サブルーチン (値を返しません)。

インターシステムズでは、プロシージャを使用することをお勧めします。それにより、変数の範囲を制御する作業がシンプルになるからです。ただし、既存のコードでは、関数およびサブルーチンも使用されているため、それらを認識できると便利です。以下のリストは、これらのすべての形式のコードがどのようなものであるかを示しています。

プロシージャ

```
label(args) scopekeyword {
    zero or more lines of code
    QUIT returnvalue
}
```

または以下のようにします。

```
label(args) scopekeyword {
    zero or more lines of code
}
```

label は、プロシージャの識別子です。

args は、オプションの引数のコンマ区切りリストです。引数がない場合でも、括弧は含める必要があります。

オプションの scopekeyword は、以下のいずれか（大文字と小文字を区別しない）です。

- ・ **Public**。Public を指定すると、そのプロシージャはパブリックとなり、そのルーチン自体の外で呼び出すことができます。
- ・ **Private**（プロシージャに対する既定値）。Private を指定すると、そのプロシージャはプライベートとなり、それと同じルーチン内の他のコードのみで呼び出すことができます。別のルーチンからそのプロシージャにアクセスしようとする、<NOLINE> エラーが発生します。

returnvalue は、オプションで返される 1 つの値です。値を返すには、QUIT コマンドを使用する必要があります。中括弧はプロシージャの終了を示すので、値を返さない場合は QUIT コマンドを省略できます。

プロシージャでは、変数をパブリック変数として宣言できます。ただし、この方法は最新の方法とは見なされていません。これを実行するには、scopekeyword の直前に、変数名をコンマで区切ったリストを角括弧で囲んで記述します。詳細は、“ObjectScript の使用法”の“[ユーザ定義コード](#)”を参照してください。

関数

```
label(args) scopekeyword
    zero or more lines of code
    QUIT optionalreturnvalue
```

args は、オプションの引数のコンマ区切りリストです。引数がない場合でも、括弧は含める必要があります。

オプションの scopekeyword は Public（関数に対する既定値）または Private です。

サブルーチン

```
label(args) scopekeyword
    zero or more lines of code
    QUIT
```

args は、オプションの引数のコンマ区切りリストです。引数がない場合、括弧の使用はオプションです。

オプションの scopekeyword は Public（サブルーチンに対する既定値）または Private です。

以下の表は、ルーチン、サブルーチン、関数、およびプロシージャの相違点を要約したものです。

	ルーチン	サブルーチン	関数	プロシージャ
引数を受け入れられるか	不可	可	可	可
値を返せるか	不可	不可	可	可
ルーチンの外で呼び出せるか（既定の場合）	可	可	可	不可
その中で定義された変数をそのコードの実行終了後に使用できるか	可	可	可	変数の特性によって異なる

詳細は、“[変数の可用性と範囲](#)”を参照してください。

注釈 一般的な使用法では、サブルーチンという用語は、プロシージャ、関数、またはサブルーチン（ここで正式に定義したものの場合）を意味することがあります。

2.14.2 ルーチンの実行

ルーチンを実行するには、以下のように DO コマンドを使用します。

ObjectScript

```
do ^routinename
```

プロシージャ、関数、またはサブルーチンを（その返り値にアクセスすることなく）実行するには、以下のコマンドを使用します。

ObjectScript

```
do label^routinename
```

または以下のようにします。

ObjectScript

```
do label^routinename(arguments)
```

プロシージャ、関数、またはサブルーチンを実行し、その返り値を参照するには、`$$label^routinename` または `$$label^routinename(arguments)` の形式の式を使用します。以下はその例です。

ObjectScript

```
set myvariable=$$label^routinename(arguments)
```

すべての場合において、ラベルが同じルーチン内にあれば、キャラットおよびルーチン名は省略できます。以下はその例です。

ObjectScript

```
do label
do label(arguments)
set myvariable=$$label(arguments)
```

すべての場合において、渡す引数は、リテラル値、式、また変数の名前です。

2.14.3 NEW コマンド

InterSystems IRIS には、ルーチンの変数の範囲を制御できるようにするもう 1 つのメカニズムがあります。NEW コマンドです。このコマンドの引数は、コンマで区切られた 1 つ以上の変数名のリストです。この変数は、パブリック変数である必要があり、グローバル変数にすることはできません。

このコマンドにより、変数 (既に存在していてもいなくてもかまいません) に対して新しい、制限されたコンテキストが確立されます。例えば、以下のルーチンを考えてみます。

ObjectScript

```
; demonew
; routine to demo NEW
NEW var2
set var1="abc"
set var2="def"
quit
```

このルーチンを実行した後は、以下のターミナル・セッションの例に示すように、変数 `var1` は使用可能であり、変数 `var2` は使用できません。

```
TESTNAMESPACE>do ^demonew

TESTNAMESPACE>write var1
abc
TESTNAMESPACE>write var2

write var2
^
<UNDEFINED> *var2
```

NEW を使用する前に変数が存在している場合は、NEW の範囲が終了した後もその変数は存在し、それが前に持っていた値を保持します。例えば、前に定義されたルーチンを使用する、以下のターミナル・セッションを考えてみます。

```
TESTNAMESPACE>set var2="hello world"

TESTNAMESPACE>do ^demonew

TESTNAMESPACE>write var2
hello world
```

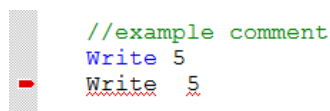
2.15 潜在的な落とし穴

以下の項目は、ObjectScript を初めて使用するプログラマ (特に、他のプログラマによって記述されたコードの管理担当者) の混乱を招くことがあります。

- ルーチンまたはメソッド内では、ラベルが含まれている行以外は、すべての行が少なくとも 1 つのスペースまたは 1 つのタブでインデントされている必要があります。つまり、最初の文字位置にテキストがある場合、コンパイラおよび IDE ではそれがラベルとして処理されます。

ただし、例外が 1 つあり、中括弧は最初の文字位置に配置できます。

- コマンドとその最初の引数の間には 1 つのスペース (タブではない) を配置する必要があります。それ以外の場合は、IDE では構文エラーがあると表示されます。



```
//example comment
Write 5
Write 5
```

同様に、ターミナルでは、以下のような構文エラーが表示されます。

```
TESTNAMESPACE>write 5
WRITE 5
^
<SYNTAX>
TESTNAMESPACE>
```

- ObjectScript で演算子の評価順序は、必ず左から右です。したがって、式の演算は表示された順番で実行されます。式で明示的に小括弧を使用して、特定の演算子を先に処理させることができます。
- 通常、括弧は、必ずしも必要でない場所にも使用します。これは、他のプログラマに（および後で作成者自身に）、コードの意図を明確にすることができるので便利です。
- これまでの使用法により、ObjectScript は空の文字列 (" ") を ASCII の NULL 値と同等とは見なしません。ASCII の NULL 値を表現するには、`$CHAR(0)` を使用します。(`$CHAR` は、ASCII 文字 (特定の 10 進数ベースのコード) を返すシステム関数です)。以下に例を示します。

ObjectScript

```
write " " = $char(0)
```

同様に、ObjectScript 値が SQL または XML に投影されるときには、値 " " と `$CHAR(0)` は異なる方法で処理されます。これらの値の SQL プロジェクションの詳細は、“InterSystems SQL の使用法”の“[NULL および空文字列](#)”を参照してください。これらの値の XML プロジェクションの詳細は、“オブジェクトの XML へのプロジェクション”の“[空文字列および NULL 値の処理](#)”を参照してください。

- ObjectScript には、大文字と小文字を区別する場合とそうでない場合があります。大文字と小文字を区別しないものには、コマンド、関数、特殊変数、ネームスペース、およびユーザの名前があります。
- 大文字と小文字を区別するものには、定義する要素の大部分（ルーチン、変数、クラス、プロパティ、およびメソッド）の名前があります。詳細は、“ObjectScript の使用法”の“[構文規則](#)”を参照してください。
- ほとんどのコマンド名は省略形で表現できます。したがって、`WRITE`、`Write`、`write`、`W`、および `w` はいずれも、`WRITE` コマンドの有効な形式です。すべての省略形は、“ObjectScript リファレンス”の“[省略形テーブル](#)”を参照してください。
- 多くのコマンドに対して、後置条件式（単に後置条件と呼ばれることが多い）を含めることができます。

この式は、InterSystems IRIS がコマンドを実行するかどうかを制御します。後置条件式の評価が `True`（ゼロ以外の値）の場合、InterSystems IRIS はそのコマンドを実行します。式の評価が `False`（ゼロ）の場合、InterSystems IRIS はコマンドを無視し、次のコマンドに実行を移します。

以下はその例です。

ObjectScript

```
Set count = 6
Write:count<5 "Print this if count is less than five"
Write:count>5 "Print this if count is greater than five"
```

上記のコードは、次の出力を生成します。Print this if count is greater than five

注釈 後置条件の使用経験がないと、“後置条件式”という表現を誤解してしまう可能性があります。この表現では、コマンドの後に式が実行されるという、誤った暗示を与えてしまうためです。その名前にかかわらず、後置条件はコマンドの前に実行されます。

- 1 つの行に複数のコマンドを含めることができます。以下はその例です。

ObjectScript

```
set myval="hello world" write myval
```

これを行うには、その行に追加のコマンドがある場合は、引数を取らないコマンドの後には必ずスペースを 2 つ使用してください。そのようにしない場合は構文エラーが発生します。

- IF、ELSE、FOR、および DO コマンドは、以下の 2 つの形式で使用できます。
 - 新しいブロック形式。中括弧を使用してブロックを示します。以下はその例です。

ObjectScript

```
if (testvalue=1) {
    write "hello world"
}
```

インターシステムズでは、新しいコードすべてでブロック形式を使用することをお勧めします。

- 古い行ベースの形式。中括弧は使用しません。以下はその例です。

ObjectScript

```
if (testvalue=1) write "hello world"
```

- 前述の項目を使用すると、ObjectScript をとてもコンパクトな形式で記述できます。以下はその例です。

ObjectScript

```
s:$g(%d(3))'=" " %d(3)=$$fdN3(%d(3)) q
```

クラス・コンパイラによって、上記のコンパクトな形式のコードが自動的に生成されます(この例のように省略された形式になるとは限りません)。場合によっては、問題の原因を追跡するため、または何かの機能のしかたを理解するために、この生成されたコードを調べると役立ちます。

- ObjectScript には予約語がないため、例えば、`set` という変数を使用することも理論的には可能です。ただし、コメント、関数、SQL 予約語、および特定のシステム項目の名前は避けた方が無難です。“ObjectScript の使用法”の[“構文規則”](#)を参照してください。
- InterSystems IRIS では、文字列操作の結果を保存するために、一定容量を持つメモリが割り当てられます。割り当てられた容量を超える文字列式が発生すると、`<MAXSTRING>` エラーとなります。上限は 3,641,144 文字です。

クラス定義の場合、文字列演算の制限は文字列プロパティのサイズに影響します。InterSystems IRIS では、この上限を超える文字列を操作する必要がある場合に使用できるシステム・オブジェクト(ストリームと呼ばれる)があります。そのような場合は、[ストリーム・インタフェース・クラス](#)を使用します。

2.16 詳細

“[クラス](#)”、“[オブジェクト](#)”、および “[永続オブジェクトと InterSystems IRIS SQL](#)” を参照してください。

3

クラス

このページでは、InterSystems IRIS® でクラスを定義および操作するための基本的な規則について説明します。

“[クラス](#)” では、オブジェクトとオブジェクト・クラスについて説明します。

3.1 クラス名とパッケージ

各 InterSystems IRIS. クラスは名前を持ち、その名前は、それが定義されている[ネームスペース](#)内で一意である必要があります。完全なクラス名は、例えば、`package.subpackage.subpackage.class` のような、1 つ以上のピリオドで区切られた文字列です。短いクラス名は、この文字列内の最後のピリオドの後の部分であり、最後のピリオドの前の部分はパッケージ名です。

パッケージ名は単なる文字列ですが、ピリオドが含まれている場合、InterSystems IRIS 開発ツールは、ピリオドで区切られた各部分をサブパッケージとして処理します。[統合開発環境 \(IDE\)](#) およびその他のツールでは、便宜上、これらのサブパッケージはフォルダの階層として表示されます。

3.2 クラス定義の基本的なコンテンツ

InterSystems IRIS クラスの定義には、以下の項目を含めることができ、これらはすべてクラス・メンバと呼ばれます。

- ・ [メソッド](#) – インスタンス・メソッドとクラス・メソッド（他の言語では静的メソッドと呼ぶ）の 2 種類のメソッドがあります。多くの場合、メソッドはサブルーチンです。
- ・ [パラメーター](#) – パラメータは、このクラスによって使用される定数値を定義します。この値はコンパイル時に設定されます。
- ・ [プロパティ](#) – プロパティには、クラスのインスタンスのデータが格納されます。
- ・ [クラス・クエリ](#) – クラス・クエリは、クラスが使用できる SQL クエリを定義し、クエリ用のコンテナとして使用するクラスを指定します。
- ・ [XData ブロック](#) – XData ブロックは、クラスによって使用される、クラス内の整形形式 XML ドキュメントです。
- ・ その他の種類のクラス・メンバ（[永続クラス](#)にのみ関連するメンバ）。

InterSystems IRIS クラス定義では、クラス定義言語 (CDL) を使用して、クラスとそのメンバを指定します。メソッド内部の実行可能コードの記述には、Python または ObjectScript を使用できます。

クラス定義には、キーワードを含めることができ、それらはクラス・コンパイラ動作に作用します。キーワードには、クラス全体に対して指定できるものと、特定のクラス・メンバに対して定義できるものがあります。これらのキーワードは、クラス・コンパイラが生成するコードに作用し、その結果、クラスの動作を制御します。

以下に、ObjectScript および Python で記述された簡単な InterSystems IRIS クラス定義とメソッドを示します。

Class/ObjectScript

```
Class MyApp.Main.SampleClass Extends %RegisteredObject
{
    Parameter CONSTANTMESSAGE [Internal] = "Hello world!" ;

    Property VariableMessage As %String [ InitialExpression = "How are you?"];
    Property MessageCount As %Numeric [Required];

    ClassMethod HelloWorld() As %String [ Language = objectscript ]
    {
        Set x=..#CONSTANTMESSAGE
        Return x
    }

    Method WriteIt() [ Language = objectscript, ServerOnly = 1]
    {
        Set count=..MessageCount
        For i=1:1:count {
            Write !,..#CONSTANTMESSAGE, " ",..VariableMessage
        }
    }
}
```

Class/Mixed

```
Class MyApp.Main.SampleClass Extends %RegisteredObject
{
    Parameter CONSTANTMESSAGE [Internal] = "Hello world!" ;

    Property VariableMessage As %String [ InitialExpression = "How are you?"];
    Property MessageCount As %Numeric [Required];

    ClassMethod MessageWrapper() As %String [ Language = objectscript ]
    {
        return ..#CONSTANTMESSAGE
    }

    ClassMethod HelloWorld() As %String [ Language = python ]
    {
        import iris
        x = iris.cls("MyApp.Main.SampleClass").MessageWrapper()
        return x
    }

    Method WriteIt() [ ServerOnly = 1, Language = python ]
    {
        import iris
        CONSTANTMESSAGE = self.MessageWrapper()
        count = self.MessageCount
        print()
        for i in range(count):
            print(CONSTANTMESSAGE, self.VariableMessage)
    }
}
```

以下の点に注意してください。

- 最初の行は、クラスの名前を指定します。MyApp.Main.SampleClass は完全なクラス名であり、MyApp.Main がパッケージ名、SampleClass が短いクラス名です。
IDE および他のユーザ・インタフェースでは、各パッケージはフォルダとして処理されます。
- Extends はコンパイラ・キーワードです。

Extends キーワードは、このクラスが **%RegisteredObject** のサブクラスであることを指定します。これは、[オブジェクトをサポート](#)するために用意されているシステム・クラスです。このサンプルのクラスでは 1 つのクラスのみを拡張していますが、他の複数のクラスを拡張することもできます。また、それらのクラスが他のクラスを拡張することもできます。

- ・ **CONSTANTMESSAGE** はパラメータです。慣例で、InterSystems IRIS システム・クラスのすべてのパラメータの名前はすべて大文字にします。これは、便利な慣例ですが、これに従う必要はありません。

Internal キーワードはコンパイラ・キーワードです。これは、このパラメータが内部であることを示し、クラス・ドキュメントで表示されないようにします。このパラメータは文字列値を持ちます。

クラス・パラメータには、ObjectScript を使用してアクセスする必要があります。このクラスの Python バージョンでは、ObjectScript クラス・メソッド `MessageWrapper()` を使用して、パラメータの値を返します。

- ・ Python からあらゆるクラス・メソッドにアクセスできます。`iris.cls("Package.Class").classMethodName()` 構文はあらゆるコンテキストで使用でき、`self.classMethodName()` 構文は Python インスタンス・メソッド内から使用できます。例では、両方の構文形式を示しています。
- ・ **VariableMessage** および **MessageCount** はプロパティです。As の後の項目は、これらのプロパティのタイプを示します。**InitialExpression** および **Required** はコンパイラ・キーワードです。

例で示しているように、InterSystems IRIS クラス・プロパティに ObjectScript または Python から直接アクセスできます。

- ・ **HelloWorld()** はクラス・メソッドであり、文字列を返します。これは As の後の項目によって示されます。

このメソッドは、クラス・パラメータの値を使用します。

- ・ **WriteIt()** はインスタンス・メソッドであり、値を返しませんが。

このメソッドは、クラス・パラメータの値と 2 つのプロパティの値を使用します。

ServerOnly コンパイラ・キーワードは、このメソッドが外部クライアントに投影されないことを意味します。

以下のターミナル・セッションは、このクラスの使用法を示しています。どちらのターミナル・シェルも、クラスの ObjectScript バージョンと Python バージョンで有効です。

ObjectScript Shell

```
TESTNAMESPACE>write ##class(MyApp.Main.SampleClass).HelloWorld()
Hello world!
TESTNAMESPACE>set x=##class(MyApp.Main.SampleClass).%New()

TESTNAMESPACE>set x.MessageCount=3

TESTNAMESPACE>do x.WriteIt()

Hello world! How are you?
Hello world! How are you?
Hello world! How are you?
```

Python Shell

```
>>> print(iris.cls("MyApp.Main.SampleClass").HelloWorld())
Hello world!
>>> x=iris.cls("MyApp.Main.SampleClass")._New()
>>> x.MessageCount=3
>>> x.WriteIt()

Hello world! How are you?
Hello world! How are you?
Hello world! How are you?
```

3.3 クラス・メソッド呼び出しのショートカット

ObjectScript を使用してクラス・メソッドを呼び出す場合、以下のシナリオではパッケージ (または上位パッケージ) を省略できます。

- ・ 参照が 1 つのクラス内であり、参照されるクラスが同じパッケージまたはサブパッケージ内にある場合。
- ・ 参照が 1 つのクラス内であり、そのクラスが `IMPORT` 指示文を使用して、参照されるクラスを含むパッケージまたはサブパッケージをインポートする場合。
- ・ 参照が 1 つのメソッド内であり、そのメソッドが `IMPORT` 指示文を使用して、参照されるクラスを含むパッケージまたはサブパッケージをインポートする場合。

ObjectScript または Python からクラス・メソッドを呼び出す場合、以下のシナリオではパッケージ (または上位パッケージ) を省略できます。

- ・ `%Library` パッケージ内のクラスを参照している場合、これは特別に処理されます。クラス `%Library.ClassName` は `%ClassName` として参照できます。例えば、`%Library.String` は `%String` として参照できます。
- ・ `User` パッケージ内のクラスを参照している場合、これは特別に処理されます。例えば、`User.MyClass` は `MyClass` として参照できます。

インターシステムズでは、`User` パッケージにはクラスを提供していません。これは、ユーザが使用するために予約されています。

他の場合はすべて、常に完全なパッケージ名とクラス名を使用してクラス・メソッドを呼び出す必要があります。

3.4 クラス・パラメータ

クラス・パラメータは、指定されたクラスのすべてのオブジェクトに対して同じ値を定義します。まれに例外がありますが、この値は、クラスのコンパイル時に確立され、実行時に変更することはできません。クラス・パラメータは、以下の目的で使用します。

- ・ 実行時に変更できない値を定義するため。
- ・ クラス定義に関するユーザ指定の情報を定義するため。クラス・パラメータは、単純な任意の名前と値の組み合わせです。この組み合わせを使用して、クラスに関する必要な情報を保存します。
- ・ プロパティとして使用する場合、さまざまなデータ型クラスの動作をカスタマイズするため (検証情報の提供など)。これについては、次のセクションで説明します。
- ・ 使用する [メソッド・ジェネレータ](#)・メソッドに対して、パラメータ化された値を提供するため。

ObjectScript メソッド、Python メソッド、またはこれら 2 つの組み合わせが含まれる InterSystems IRIS クラスのパラメータを定義できます。以下に、いくつかのパラメータを持つクラスを示します。

Class Definition

```
Class GSOP.DivideWS Extends %SOAP.WebService
{
Parameter USECLASSNAMESPACES = 1;

/// Name of the Web service.
Parameter SERVICENAME = "Divide";

/// SOAP namespace for the Web service
Parameter NAMESPACE = "http://www.mynamespace.org";

/// let this Web service understand only SOAP 1.2
Parameter SOAPVERSION = "1.2";

///further details omitted
}
```

注釈 クラス・パラメータは式としても指定できます。その式は、コンパイル時か実行時に評価できます。詳細は、“[クラス・パラメータの定義と参照](#)”を参照してください。

3.5 プロパティ

InterSystems IRIS には、以下の 2 種類のプロパティがあります。

- ・ 属性 - 値を保持します。この値は、以下のいずれかです。
 - 1 つのリテラル値。通常、データ型に基づきます。
 - [オブジェクト](#)値 (コレクション・オブジェクトとストリーム・オブジェクトがあります)。
 - 多次元配列。これはあまり一般的ではありません。

プロパティという用語は、関連付けを保持するプロパティではなく、単に、属性のプロパティを示すことがよくあります。

- ・ リレーションシップ - オブジェクト間の関連付けを保持します。

ObjectScript メソッド、Python メソッド、またはこれら 2 つの組み合わせが含まれるクラスのプロパティを定義できます。ただし、Python メソッドからリレーションシップにアクセスすることはできません。このセクションでは、これらのバリエーションのいくつかを示すプロパティ定義を含むサンプル・クラスを示します。

Class Definition

```
Class MyApp.Main.Patient Extends %Persistent
{
Property PatientID As %String [Required];
Property Gender As %String(DISPLAYLIST = ",Female,Male", VALUELIST = ",F,M");
Property BirthDate As %Date;
Property Age As %Numeric [Transient];
Property MyTempArray [MultiDimensional];
Property PrimaryCarePhysician As Doctor;
Property Allergies As list Of PatientAllergy;
Relationship Diagnoses As PatientDiagnosis [ Cardinality = children, Inverse = Patient ];
}
```

以下のことに注意してください。

- それぞれの定義では、`As` の後の項目はプロパティのタイプです。各タイプはクラスです。構文 `As List Of` は、特定のコレクション・クラスの省略表現です。
`%String`、`%Date`、および `%Numeric` はデータ型クラスです。
`%String` は既定のタイプです。
- `Diagnoses` は、リレーションシップ・プロパティであり、残りは属性プロパティです。
- `PatientID`、`Gender`、`BirthDate`、および `Age` は、単純なリテラル値のみを格納できます。
- `PatientID` は `Required` キーワードを使用するため、必須です。つまり、このクラスのオブジェクトは、このプロパティの値を指定しない場合、保存できません。
- `Age` は、他のリテラル・プロパティとは異なり、ディスクに保存されません。これは、それが `Transient` キーワードを使用するためです。
- `MyTempArray` は `MultiDimensional` キーワードを使用するため、多次元プロパティです。このプロパティは既定では、ディスクに保存されません。
- `PrimaryCarePhysician` および `Allergies` は、オブジェクト値プロパティです。
- `Gender` プロパティ定義には、プロパティ・パラメータの値が含まれます。これらは、このプロパティが使用するデータ型クラスのパラメータです。
このプロパティの値は、`M` と `F` に制限されます。表示値を表示すると (管理ポータルなどで)、代わりに `Male` と `Female` が表示されます。各データ型クラスでは、`LogicalToDisplay()` などのメソッドが提供されます。

3.5.1 プロパティ・キーワードの指定

プロパティ定義では、プロパティの使用法に作用するオプションのプロパティ・キーワードを含めることができます。以下のリストに、よく使用されているキーワードのいくつかを示します。

Required

このクラスのインスタンスをディスクに格納する前に、プロパティの値を設定する必要があることを指定します。既定では、プロパティは `Required` ではありません。サブクラスで、必要に応じてオプションのプロパティをマークできますが、その逆は実行できません。

InitialExpression

プロパティの初期値を指定します。既定では、プロパティは初期値を持ちません。サブクラスは `InitialExpression` キーワードの値を継承し、それをオーバーライドできます。指定する値は、有効な `ObjectScript` 式である必要があります。

Transient

プロパティをデータベースに格納しないことを指定します。既定では、プロパティは `Transient` ではありません。サブクラスは `Transient` キーワードの値を継承しますが、それをオーバーライドできません。

Private

プロパティがプライベートであることを指定します。サブクラスは `Private` キーワードの値を継承しますが、それをオーバーライドできません。

既定では、プロパティはパブリックであり、どのような場所でもアクセスできます。プロパティは (`Private` キーワードを使用して) プライベートとしてマークできます。その場合、それらが属するオブジェクトのメソッドによってのみアクセスできます。

InterSystems IRIS では、プライベート・プロパティは常に、プロパティを定義したクラスのサブクラスに継承され、参照できます。

他のプログラミング言語では、これらを保護されたプロパティと呼ぶことがあります。

Calculated

このプロパティには、メモリ内ストレージを割り当てないことを指定します。既定では、プロパティは Calculated ではありません。サブクラスは Calculated キーワードを継承しますが、それをオーバーライドできません。

MultiDimensional

プロパティが多次元であることを指定します。このプロパティは、以下の点で他のプロパティとは異なります。

- ・ 関連付けられているメソッドがありません（後続のトピックを参照）。
- ・ オブジェクトを検証または保存するときに、無視されます。
- ・ アプリケーションにそれを明示的に保存するコードが含まれていない場合、ディスクに保存されません。
- ・ クライアント・テクノロジーに公開することはできません。
- ・ SQL テーブルに格納できず、SQL テーブルでも公開されません。

多次元プロパティはあまり使用されませんが、オブジェクト状態情報を一時的に格納する場合に役立ちます。

3.6 データ型に基づくプロパティ

プロパティを定義し、そのタイプをデータ型クラスと指定する場合、このセクションで説明するように、そのプロパティを定義し、操作するための特別なオプションがあります。

3.6.1 データ型クラス

データ型クラスでは、プロパティの値に関する一連のルールを適用できます。

InterSystems IRIS には、`%Library.String`、`%Library.Integer`、`%Library.Numeric`、`%Library.Date` など多くのデータ型クラスが用意されています。`%Library` パッケージのクラスの名前は省略可能なため、これらの多くを省略できます。例えば、`%Date` は `%Library.Date` の省略形です。

各データ型クラスには、以下の機能があります。

- ・ コンパイラ・キーワードの値を指定します。プロパティの場合、コンパイラ・キーワードは、以下のようなことを実行できます。
 - プロパティを必須にします。
 - プロパティの初期値を指定します。
 - プロパティを SQL、ODBC、および Java クライアントに投影する方法を制御します。
- ・ 以下のような詳細に影響を与えるパラメータの値を指定します。
 - データ型の論理値の最大値と最小値
 - 文字列の最大文字数と最小文字数
 - 小数点以下の桁数
 - 最大文字数を超えた場合に文字列を切り捨てるかどうか

- 表示形式
 - 特別な XML または HTML 文字のエスケープ方法
 - ユーザ・インタフェースで使用するための論理値および表示値の列举リスト
 - 文字列が一致する必要があるパターン (InterSystems IRIS パターン・マッチング演算子を自動的に使用)
 - XML のインポートまたは XML へのエクスポートの際に UTC タイム・ゾーンを尊重するか、無視するか
- これは、リテラル・データを、格納形式 (ディスク上)、論理形式 (メモリ内)、表示形式の間で変換するための一連のメソッドを提供します。

独自のデータ型クラスを追加できます。例えば、以下は `%Library.String` のカスタム・サブクラスを示しています。

Class Definition

```
Class MyApp.MyType Extends %Library.String
{
    /// The maximum number of characters the string can contain.
    Parameter MAXLEN As INTEGER = 2000;
}
```

3.6.2 データ型クラスのパラメータのオーバーライド

プロパティを定義し、そのタイプをデータ型クラスと指定する場合、そのデータ型クラスで定義されるどのパラメータもオーバーライドできます。

例えば、`%Integer` データ型クラスは、クラス・パラメータ (MAXVAL) を定義しますが、このパラメータに対して値を提供しません。これを、プロパティ定義で以下のようにオーバーライドできます。

Class Member

```
Property MyInteger As %Integer(MAXVAL=10);
```

このプロパティの場合、許容最大値は 10 です。

(データ型クラスの検証メソッドはメソッド・ジェネレータであるため、これは内部で機能します。指定するパラメータ値は、コンパイラがそのクラスのコードを生成するときに使用されます。

同様に、タイプ `%String` のすべてのプロパティに照合タイプがあります。これにより、値を並べる方法 (先頭の文字の大文字化が有効かどうかなど) が決まります。既定の照合タイプは、SQLUPPERです。

以下の別の例では、データ型クラスは、DISPLAYLIST および VALUelist パラメータを定義します。これらを使用して、ユーザ・インタフェースに表示する選択肢と、それらに対応する内部値を指定できます。

Class Member

```
Property Gender As %String(DISPLAYLIST = ",Female,Male", VALUelist = ",F,M");
```

3.6.3 他のプロパティ・メソッドの使用法

プロパティには、自動的に関連付けられた多くのメソッドがあります。これらのメソッドは、データ型クラスによって生成され、ObjectScript からアクセスできます。

例えば、3 つのプロパティでクラス `Person` を定義する場合、以下のようになります。

Class Definition

```
Class MyApp.Person Extends %Persistent
{
  Property Name As %String;
  Property Age As %Integer;
  Property DOB As %Date;
}
```

生成された各メソッドの名前は、プロパティ名に継承クラスからのメソッド名を連結した名前です。以下の例に示すように、生成されたこれらのメソッドに ObjectScript からアクセスできます。Python から同じ情報にアクセスするには、関連付けられているメソッドを継承クラスから直接呼び出します。例えば、**%Date** クラスに関連付けられている一部のメソッド、したがって **DOB** プロパティでは以下のようになります。

ObjectScript

```
Set x = person.DOBIsValid(person.DOB)
Write person.DOBLogicalToDisplay(person.DOB)
```

Python

```
x = iris.cls("%Date").IsValid(person.DOB)
print(iris.cls("%Date").LogicalToDisplay(person.DOB))
```

IsValid はプロパティ・クラスのメソッドで、LogicalToDisplay は **%Date** データ型クラスのメソッドです。

3.7 メソッド

インスタンス・メソッドとクラス・メソッド (他の言語では静的メソッドと呼ぶ) の 2 種類のメソッドがあります。

3.7.1 Method キーワードの指定

メソッド定義では、メソッドがどのように動作するかに作用するオプションのコンパイラ・キーワードを含めることができます。以下のリストに、よく使用されているメソッド・キーワードのいくつかを示します。

Language

InterSystems IRIS では、メソッドを ObjectScript または Python で記述できます。メソッドの記述に使用する言語を指定するには、以下の構文を使用します。

Method/ObjectScript

```
Method MyMethod() [ Language = objectscript ]
{
  // implementation details written in ObjectScript
}
```

Method/Python

```
Method MyMethod() [ Language = python ]
{
  # implementation details written in Python
}
```

メソッドで Language キーワードを使用しない場合、コンパイラはメソッドが ObjectScript で記述されていると想定します。

前述の例のように、メソッドの言語はすべて小文字で記述する必要があります。

Private

このキーワードはメソッドがプライベートであることを指定するもので、ObjectScript メソッドでのみ使用できます。サブクラスは Private キーワードの値を継承しますが、それをオーバーライドできません。

既定では、メソッドはパブリックであり、どのような場所でもアクセスできます。メソッドは (Private キーワードを使用して) プライベートとしてマークできます。その場合、

- ・ それらが属するクラスのメソッドによってのみアクセスできます。
- ・ インターシステムズ・クラス・リファレンスには記載されません。

ただし、これは継承され、このメソッドを定義するクラスのサブクラスで使用できます。

他の言語では、このようなメソッドを保護されたメソッドと呼ぶことがあります。

3.7.2 他のクラス・メンバの参照

メソッド内で、ここに示す構文を使用し、他のクラスのメンバを参照します。

- ・ ObjectScript でパラメータを参照するには、以下のような式を使用します。

ObjectScript

```
..#PARAMETERNAME
```

パラメータに直接アクセスできるのは、ObjectScript を使用する場合のみです。Python からパラメータにアクセスするには、ObjectScript ラップ・メソッドを使用してそのパラメータを返し、必要に応じてこのメソッドを呼び出します。以下に例を示します。

Class/Mixed

```
Class User.Employee Extends %RegisteredObject
{
    Parameter ADDRESS = "123 Main St.";

    ClassMethod AddressWrapper() As %String [ Language = objectscript ]
    {
        return ..#ADDRESS
    }

    ClassMethod OfficeLocation() [ Language = python ]
    {
        import iris
        location=iris.cls("User.Employee").AddressWrapper()
        print("This office is located at", location)
    }

    Method EmployeeLocation() [ Language = python ]
    {
        location=self.AddressWrapper()
        print("This employee works at", location)
    }
}
```

Python からクラス・メソッドにアクセスする場合、あらゆるコンテキストで `iris.cls("Package.Class").classMethodName()` 構文を使用できます。Python インスタンス・メソッド内からであれば、短い `self.classMethodName()` 構文を使用することもできます。

インターシステムズが提供するクラスでは、慣例ですべてのパラメータがすべて大文字で定義されていますが、作成するコードでは大文字にしなくてもかまいません。

- ・ 別のインスタンス・メソッドを参照するには、以下のような式を使用します。

ObjectScript

```
..methodname(arguments)
```

Python

```
self.methodname(arguments)
```

この構文をクラス・メソッド内で使用して、インスタンス・メソッドを参照することはできません。

- ・ 別のクラス・メソッドを参照するには、以下の構文を使用します。

ObjectScript

```
..classmethodname(arguments)
```

Python

```
iris.cls("Package.Class").classmethodname(arguments)
```

Python の self 構文を使用してクラス・メソッドにアクセスすることはできません。

- ・ (インスタンス・メソッド内のみ) インスタンスのプロパティを参照するには、以下のような式を使用します。

ObjectScript

```
..PropertyName
```

Python

```
self.PropertyName
```

同様に、オブジェクト値プロパティのプロパティを参照するには、以下のような式を使用します。

ObjectScript

```
..PropertyNameA.PropertyNameB
```

Python

```
self.PropertyNameA.PropertyNameB
```

ObjectScript の例で使用されている構文は、InterSystems IRIS ドット構文と呼ばれます。

また、オブジェクト値プロパティのインスタンス・メソッドまたはクラス・メソッドを呼び出すこともできます。以下に例を示します。

ObjectScript

```
Do ..PropertyName.MyMethod()
```

Python

```
self.PropertyName.MyMethod()
```

3.7.3 他のクラスのメソッドの参照

メソッド内で (またはルーチン内で)、ここに示す構文を使用し、他のクラスのメソッドを参照します。

- ・ クラス・メソッドを呼び出し、その返り値にアクセスするには、以下のような式を使用します。

ObjectScript

```
##class(Package.Class).MethodName(arguments)
```

Python

```
iris.cls("Package.Class").MethodName(arguments)
```

以下はその例です。

ObjectScript

```
Set x=##class(Util.Utils).GetToday()
```

Python

```
x=iris.cls("Util.Utils").GetToday()
```

以下のように、戻り値にアクセスせずにクラス・メソッドを呼び出すこともできます。

ObjectScript

```
Do ##class(Util.Utils).DumpValues()
```

Python

```
iris.cls("Util.Utils").DumpValues()
```

注釈 ##class は、大文字と小文字を区別しません。

- ・ インスタンス・メソッドを呼び出すには、[インスタンス](#)を作成し、ObjectScript または Python で以下のような式を使用してそのメソッドを呼び出し、その戻り値にアクセスします。

```
instance.MethodName(arguments)
```

以下に例を示します。

ObjectScript

```
Set x=instance.GetName()
```

Python

```
x=instance.GetName()
```

以下のように呼び出すことで、戻り値にアクセスせずにインスタンス・メソッドを呼び出すこともできます。

ObjectScript

```
Do instance.InsertItem("abc")
```

Python

```
instance.InsertItem("abc")
```

返り値がないメソッドもあるので、自身の状況にあった構文を選択してください。

3.7.4 現在のインスタンスの参照

場合によっては、インスタンス・メソッド内で、そのインスタンスのプロパティまたはメソッドではなく、現在のインスタンス自体を参照することが必要になります。例えば、他のコードを呼び出すときに、現在のインスタンスを渡すことが必要な場合があります。

ObjectScript では、特殊変数 \$THIS を使用して現在のインスタンスを参照します。Python では、変数 self を使用して現在のインスタンスを参照します。

以下に例を示します。

ObjectScript

```
Set sc=header.ProcessService($this)
```

Python

```
sc=header.ProcessService(self)
```

3.7.5 メソッド引数

メソッドは、コンマ区切りリストで位置を示す引数を取ります。引数ごとに、型および既定の値を指定できます。

例えば、以下は、3 つの引数を取るメソッドの定義の一部分です。これは、InterSystems IRIS クラス内の ObjectScript メソッドと Python メソッドの両方で有効な構文です。

Class Member

```
Method Calculate(count As %Integer, name, state As %String = "CA") as %Numeric
{
    // ...
}
```

引数のうち 2 つが明示的な型を持ち、1 つが既定の値を持っています。一般的に、各引数の型を明示的に指定することは良い方法です。

注釈 メソッドが Python で定義されていて、そのメソッドの引数に既定値が指定されている場合、その引数を引数リストの末尾に配置して、コンパイル・エラーを避ける必要があります。

3.7.5.1 引数のスキップ

メソッドに適切な既定値がある場合、メソッドを呼び出すときに引数をスキップできます。ObjectScript と Python にはそれぞれ、引数をスキップするための独自の構文があります。

ObjectScript で引数をスキップするには、引数の値を指定せずにコンマ構造を維持します。例えば、以下は有効です。

ObjectScript

```
set myval=##class(mypackage.myclass).GetValue(,,,,,4)
```

InterSystems IRIS クラスでは、Python メソッドのシグニチャで最初に必須の引数を記述し、その後に引数と既定値を記述する必要があります。

そのメソッドを呼び出す際に、引数をメソッドのシグニチャの順番で指定する必要があります。したがって、1 つの引数をスキップした場合、その後の引数もすべてスキップする必要があります。例えば、以下は有効です。

Member/Python

```
ClassMethod Skip(a1, a2 As %Integer = 2, a3 As %Integer = 3) [ Language = python ]
{
    print(a1, a2, a3)
}

TESTNAMESPACE>do ##class(mypackage.myclass).Skip(1)
1 2 3
```

3.7.5.2 値または参照による変数渡し

メソッドを呼び出す場合、値または参照によって、そのメソッドに変数の値を渡すことができます。

通常、メソッドのシグニチャは、引数の参照渡しを意図しているかどうかを示します。以下はその例です。

```
Method MyMethod(argument1, ByRef argument2, Output argument3)
```

ByRef キーワードは、参照によってこの引数を渡すことを示します。Output キーワードは、この引数を参照によって渡すことと、この引数に最初に指定した値がメソッドによってすべて無視されることを示します。

同様に、メソッドを定義するときに、メソッド・シグニチャで ByRef および Output キーワードを使用すると、そのメソッドの他のユーザに、それをどのように使用することを意図しているのかを知らせることができます。

ObjectScript で引数を参照渡しするには、そのメソッドを呼び出すときに変数名の前にピリオドを付けます。Python では、渡す値に対して `iris.ref()` を使用し、その参照に対してメソッドを呼び出します。これら両方を以下の例に示します。

ObjectScript

```
Do MyMethod(arg1, .arg2, .arg3)
```

Python

```
arg2=iris.ref("peanut butter")
arg3=iris.ref("jelly")
MyMethod(arg1,arg2,arg3)
```

重要 ByRef および Output キーワードは、インターシステムズ・クラス・リファレンスの利用者全員に役立つ情報を提供します。これらのキーワードは、コードの動作には影響しません。メソッドの呼び出し方法に関する規則を適用するのは、メソッドの作成者の責任です。

3.7.5.3 可変の引数

引数の個数を変えることができるメソッドを定義できます。以下はその例です。

Method/ObjectScript

```
ClassMethod MultiArg(Arg1... As %List) [ Language = objectscript ]
{
    Set args = $GET(Arg1, 0)
    Write "Invocation has ",
        args,
        " element",
        $SELECT((args=1):"", 1:"s"), !
    For i = 1 : 1 : args
    {
        Write "Argument[" , i , "]: " , $GET(Arg1(i), "<NULL>"), !
    }
}
```

Method/Python

```
ClassMethod MultiArg(Arg1... As %List) [ Language = Python ]
{
    print("Invocation has", len(Arg1), "elements")
    for i in range(len(Arg1)):
        print("Argument[" + str(i+1) + "]: " + Arg1[i])
}
```

3.7.5.4 既定値の指定

ObjectScript または Python メソッドで引数の既定値を指定するには、以下の例に示す構文を使用します。

Class Member

```
Method Test(flag As %Integer = 0)
{
    //method details
}
```

メソッドが呼び出されるとき、引数が指定されていない場合は既定値を使用します（既定値が指定されている場合）。メソッドが Python で記述されている場合、既定値を持つ引数は、引数リストの末尾に定義する必要があります。

ObjectScript では、\$GET 関数を使用して既定値を設定することもできます。以下はその例です。

Class Member

```
Method Test(flag As %Integer)
{
    set flag=$GET(flag,0)
    //method details
}
```

ただし、この方法ではシグニチャに影響しません。

3.8 メソッド・ジェネレータ

メソッド・ジェネレータは、クラスのコンパイル中にクラス・コンパイラによって呼び出されるプログラムです。この出力は、メソッドの実際の実行時実装です。メソッド・ジェネレータは強力なクラス継承の方法を提供し、クラスやプロパティ継承の必要性に応じてカスタマイズされた、高性能で特別なコードを生成します。InterSystems IRIS ライブラリ内で、メソッド・ジェネレータは、データ型やストレージ・クラスによって広範囲に使用されます。

3.9 クラス・クエリ

InterSystems IRIS クラスには、クラス・クエリを含めることができます。クラス・クエリは、クラスが使用できる SQL クエリを定義し、クエリのためのコンテナとして使用するクラスを指定します。以下に例を示します。

Class Member

```
Query QueryName(Parameter As %String) As %SQLQuery
{
    SELECT MyProperty, MyOtherProperty FROM MyClass
    WHERE (MyProperty = "Hello" AND MyOtherProperty = :Parameter)
    ORDER BY MyProperty
}
```

アプリケーションで使用するための事前に定義された検索を提供するために、クラス・クエリを定義します。例えば、名前などの複数のプロパティによってインスタンスを検索したり、パリからマドリッドまでのすべての航空便など、一連の特定の

条件に適合するインスタンスのリストを提供したりできます。ここに示した例では、パラメータを使用します。これは、柔軟なクエリを提供するための一般的な方法です。クラス・クエリはどのクラス内でも定義できます。クラス・クエリは永続クラス内に含める必要はありません。これについては、このドキュメントで後述します。

3.10 XData ブロック

XML は、多くの場合、構造化されたデータを表すための便利な方法であるため、InterSystems IRIS クラスには、どのようなニーズに対しても整形 XML ドキュメントを含めることを可能にするメカニズムがあります。このためには、もう 1 つの種類のクラス・メンバである XData ブロックを含めます。

InterSystems IRIS は、特定の目的のために XData ブロックを使用します。場合によっては、これらをユーザ自身のアプリケーションに活用することができます。

- ・ InterSystems IRIS Web サービスおよび Web クライアントに対する WS-Policy サポート。“[Web サービスおよび Web クライアントの作成](#)”を参照してください。この場合、XData ブロックはセキュリティ・ポリシーを記述します。
- ・ Business Intelligence では、XData ブロックを使用してキューブ、サブジェクト領域、KPI、およびその他の要素を定義します。

詳細は、“[XData ブロックの定義と使用](#)”を参照してください。

3.11 クラス定義におけるマクロとインクルード・ファイル

InterSystems IRIS クラス定義では、ObjectScript メソッド内にマクロを定義し、それらのマクロをそのメソッド内で使用できます。ただし、多くの場合、それらを[インクルード・ファイル](#)内で定義し、それをクラス定義の先頭でインクルードできます。以下はその例です。

```
Include (%assert, %callout, %occInclude, %occSAX)

/// Implements an interface to the XSLT Parser. XML contained in a file or binary
/// stream may be transformed
Class %XML.XSLT.Transformer Extends %RegisteredObject ...
```

その後、そのクラスの ObjectScript メソッドは、そのインクルード・ファイル、またはそれにインクルードされたインクルード・ファイルに定義されたマクロを参照できます。

マクロは継承されます。つまり、サブクラスは、そのスーパークラスと同じマクロすべてにアクセスできます。

3.12 InterSystems IRIS における継承規則

他のクラスベースの言語と同様に、複数のクラス定義を継承によって結合できます。InterSystems IRIS クラス定義は、複数の他のクラスを拡張（または他のクラスから継承）できます。また、それらのクラスが他のクラスを拡張することもできます。

InterSystems IRIS クラスは、Python で定義されたクラス（つまり、.py ファイルに含まれるクラス定義）から継承することはできません（逆の場合も同様です）。

以下のサブセクションでは、InterSystems IRIS のクラスにおける継承の基本規則について説明します。

3.12.1 継承順序

InterSystems IRIS では、継承順序に次の規則を使用します。

1. 既定では、特定の名前のクラス・メンバが複数のスーパークラスで定義されている場合、そのサブクラスはスーパークラス・リストの左端のクラスから定義を取ります。
2. クラス定義に `Inheritance = right` が含まれている場合、サブクラスはスーパークラス・リストの右端のクラスから定義を取ります。

これまでの使用法により、大部分の InterSystems IRIS クラスには、`Inheritance = right` が含まれています。

3.12.2 プライマリ・スーパークラス

他のクラスを拡張するクラスはすべて、1 つのプライマリ・スーパークラスを持っています。

クラスで使用される継承順序に関係なく、プライマリ・スーパークラスは、左から右に読む場合の最初のクラスです。

どのクラス・レベルのコンパイラ・キーワードについても、指定されたクラスは、そのプライマリ・スーパークラスで指定された値を使用します。

永続クラスの場合、プライマリ・スーパークラスは特に重要です。“[クラスとエクステント](#)”を参照してください。

3.12.3 最も適切なタイプのクラス

あるオブジェクトが複数のクラスのエクステント（いくつかのスーパークラスのエクステントなど）に属するインスタンスである場合でも、そのオブジェクトには必ず最も適切なタイプのクラス（MSTC）があります。オブジェクトがクラスのインスタンスであるが、そのクラスのどのサブクラスのインスタンスにもなっていない場合、そのクラスはそのオブジェクトの最も適切なタイプのクラスになります。

3.12.4 メソッドのオーバーライド

クラスは、その 1 つまたは複数のスーパークラスからメソッド（クラスおよびインスタンス・メソッドの両方）を継承し、それらはオーバーライドできます。その場合、自身のメソッド定義のシグニチャが、オーバーライドするメソッドのシグニチャと一致していることを確認する必要があります。サブクラス・メソッドの各引数は、スーパークラス・メソッドの引数と同じデータ型を使用するか、そのデータ型のサブクラスを使用する必要があります。ただし、サブクラスのメソッドで、そのスーパークラスに定義されていない追加の引数を指定することはできません。

メソッドのシグニチャが一致する限り、ObjectScript で記述されているメソッドを Python メソッドでオーバーライドすることも、その逆を行うこともできます。

サブクラスのメソッド内で、スーパークラス内のそれがオーバーライドしたメソッドを参照できます。ObjectScript でそれを行うには、`##super()` 構文を使用します。以下はその例です。

```
//overrides method inherited from a superclass
Method MyMethod() [ Language = objectscript ]
{
    //execute MyMethod as implemented in the superclass
    do ##super()
    //do more things....
}
```

注釈 `##super` は、大文字と小文字を区別しません。

3.13 詳細

これらのトピックの詳細は、以下のリソースを参照してください。

- ・ [“クラスの定義と使用”](#) では、InterSystems IRIS でのクラスおよびクラス・メンバの定義方法について説明しています。
- ・ [“クラス定義リファレンス”](#) には、クラス定義で使用するコンパイラ・キーワードのリファレンス情報があります。
- ・ [“インターシステムズ・クラス・リファレンス”](#) には、InterSystems IRIS で提供されるすべての非内部クラスに関する詳細があります。

4

オブジェクト

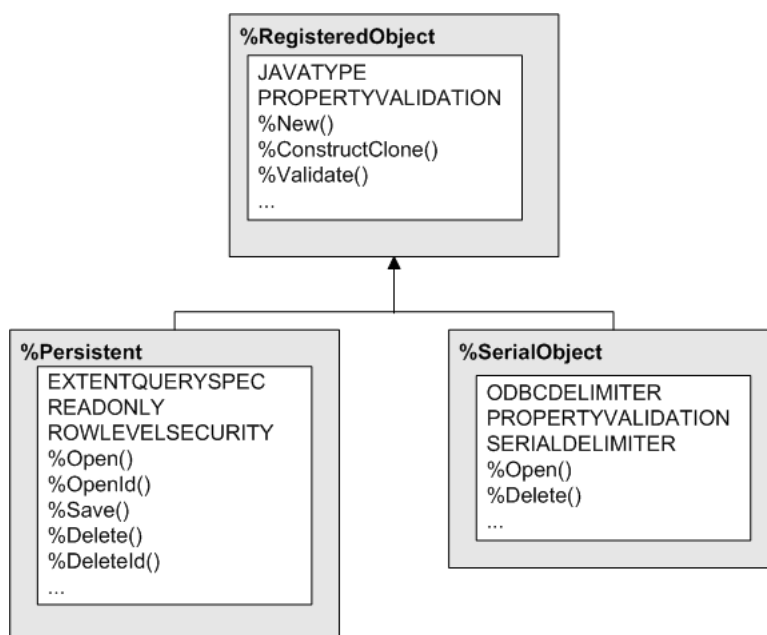
このページでは、InterSystems IRIS® のオブジェクトについて説明します。

このページで示されているコード・サンプルでは、Samples-Data サンプル (<https://github.com/interSystems/Samples-Data>) のクラスを使用しています。(例えば) **SAMPLES** という名前の専用ネームスペースを作成し、そのネームスペースにサンプルをロードすることをお勧めします。一般的な手順は、“[InterSystems IRIS で使用するサンプルのダウンロード](#)”を参照してください。

4.1 InterSystems IRIS オブジェクト・クラスの概要

InterSystems IRIS は、**%Library.RegisteredObject**、**%Library.Persistent**、および **%Library.SerialObject** のオブジェクト・クラスを使用してオブジェクト・テクノロジーを提供します。

以下の図は、これらのクラス間の継承関係、およびそれらのパラメータとメソッドのいくつかを示しています。**%Library** パッケージのクラスの名前は省略可能です。したがって、例えば、**%Persistent** は **%Library.Persistent** の省略形です。ここで、すべて大文字の項目はパラメータであり、パーセント記号で始まる項目はメソッドです。



通常のクラスベースのアプリケーションでは、これらのクラス（および専用のシステム・サブクラス）をベースにしてクラスを定義します。すべてのオブジェクトは、これらのクラスの 1 つから直接または間接継承し、すべてのオブジェクトは、以下のタイプのいずれかです。

- ・ 登録オブジェクトは、**%RegisteredObject** またはサブクラスのインスタンスです。これらのオブジェクトは作成できますが、保存することはできません。他の 2 つのクラスは **%RegisteredObject** から継承し、したがって、そのクラスのすべてのパラメータ、メソッドなどを含みます。
- ・ 永続オブジェクトは、**%Persistent** またはサブクラスのインスタンスです。これらのオブジェクトは、作成、保存、オープン、および削除できます。

永続クラスは、InterSystems SQL を使用してアクセスできるテーブルへ自動的に投影されます。

- ・ シリアル・オブジェクトは、**%SerialObject** またはサブクラスのインスタンスです。シリアル・クラスは、別のオブジェクトのプロパティとして使用するためのものです。これらのオブジェクトを作成することはできますが、オブジェクトを保存したり、それらを含むオブジェクトとは別に単独で開いたりすることはできません。

これらのオブジェクトは、永続オブジェクトに含まれている場合、SQL に自動的に投影されます。

注釈 **%DynamicObject** クラスと **%DynamicArray** クラスを使用することで、InterSystems IRIS では、スキーマを持たないオブジェクトと配列の操作もできます。これらのクラスについては、このドキュメントでは説明しません。詳細は、“[JSON の使用](#)”を参照してください。

4.2 オブジェクト・クラスの基本機能

オブジェクト・クラスを使用すると、以下のタスクを実行できます。

- ・ オブジェクト（クラスのインスタンス）を作成できます。このためには、そのクラスの **%New()** メソッドを使用します。これは、**%RegisteredObject** から継承します。

以下はその例です。

ObjectScript

```
set myobj=##class(Sample.Person).%New()
```

Python

```
myobj = iris.cls("Sample.Person")._New()
```

Python メソッド名ではパーセント記号 (%) を使用できません。% 文字が含まれる ObjectScript メソッドを Python から呼び出すには、上の例のように % 文字をアンダースコア () に置き換えます。

- ・ プロパティを使用できます。

どのクラスでもプロパティを定義できますが、インスタンスを作成できるのはオブジェクト・クラスのみであるため、プロパティはオブジェクト・クラスでのみ役立ちます。

どのプロパティにも 1 つのリテラル値、オブジェクト（コレクション・オブジェクトの場合がある）、または多次元配列（この場合はほとんどない）が含まれます。以下は、オブジェクト値プロパティの定義の例です。

Class Member

```
Property Home As Sample.Address;
```

Sample.Address は別のクラスです。以下に、Home プロパティの値を設定する 1 つの方法を示します。

ObjectScript

```
Set myaddress=##class(Sample.Address).%New()
Set myaddress.City="Louisville"
Set myaddress.Street="15 Winding Way"
Set myaddress.State="Georgia"

Set myperson=##class(Sample.Person).%New()
Set myperson.Home=myaddress
```

Python

```
import iris
myaddress=iris.cls("Sample.Address")._New()
myaddress.City="Louisville"
myaddress.Street="15 Winding Way"
myaddress.State="Georgia"

myperson=iris.cls("Sample.Person")._New()
myperson.Home=myaddress
```

- そのクラスまたはそのスーパークラスがインスタンス・メソッドを定義する場合、そのクラスのインスタンスのメソッドを呼び出すことができます。以下はその例です。

Method/ObjectScript

```
Method PrintPerson() [ Language = objectscript ]
{
    Write !, "Name: ", ..Name
}
```

Method/Python

```
Method PrintPerson() [ Language = python ]
{
    print("\nName:", self.Name)
}
```

myobj がこのメソッドを定義するクラスのインスタンスである場合、このメソッドを以下のように呼び出すことができます。

ObjectScript

```
Do myobj.PrintPerson()
```

Python

```
myobj.PrintPerson()
```

- プロパティ値が、そのプロパティ定義で指定されている規則に従っていることを検証できます。
 - すべてのオブジェクトが、そのオブジェクトのすべてのプロパティ値を正規化するインスタンス・メソッド %NormalizeObject() を継承します。多くのデータ型により、同じ値を異なる表現にすることができます。正規化により、値が、そのキャノニック形式、つまり正規化された形式に変換されます。%NormalizeObject() は、この処理が正常に実行されたかどうかによって True または False を返します。
 - すべてのオブジェクトは、インスタンス・メソッド %ValidateObject() を継承し、このメソッドは、プロパティ値がプロパティ定義に従っているかどうかに応じて True または False を返します。
 - すべての永続オブジェクトは、インスタンス・メソッド %Save() を継承します。%Save() インスタンス・メソッドを使用する場合、システムによって %ValidateObject() が最初に呼び出されます。

対照的に、ルーチン・レベルで作業していて、クラスを使用しない場合、コードに、型とその他の入力要件をチェックするロジックが含まれている必要があります。

- コールバック・メソッドを定義して、オブジェクトを作成、変更などするときの追加のカスタムの動作を追加できます。

例えば、クラスのインスタンスを作成するには、そのクラスの %New() メソッドを呼び出します。そのクラスで %OnNew() メソッド (コールバック・メソッド) が定義されている場合、InterSystems IRIS によってそのメソッドも自動的に呼び出されます。以下に簡単な例を示します。

Method/ObjectScript

```
Method %OnNew() As %Status
{
    Write "hi there"
    Return $$$OK
}
```

Method/Python

```
Method %OnNew() As %Status [ Language = python ]
{
    print("hi there")
    return True
}
```

実際のシナリオでは、このコールバックによっていくつかの必須の初期化が実行される場合があります。また、ファイル、または場合によってはグローバルへの書き込みによるログが実行されることもあります。

4.3 OREF

オブジェクト・クラスの %New() メソッドは、オブジェクトのデータを格納するための内部のメモリ内の構造を作成し、その構造を指す OREF (オブジェクト参照) を返します。OREF は、InterSystems IRIS の特別な種類の値です。以下の点に留意してください。

- ・ ターミナルでは、OREF の内容は使用する言語によって異なります。
 - ObjectScript では、数字、次にアット記号 (@)、その次にクラスの名前が続いた文字列が表示されます。
 - Python では、クラス名と、メモリ内の固有の場所を示す 18 文字を含む文字列が表示されます。

以下に例を示します。

ObjectScript Shell

```
TESTNAMESPACE>set myobj=##class(Sample.Person).%New()
TESTNAMESPACE>w myobj
3@Sample.Person
```

Python Shell

```
>>> myobj=iris.cls("Sample.Person")._New()
>>> print(myobj)
<iris.Sample.Person object at 0x000001A1E52FFD20>
```

- ・ OREF が必要な場所で OREF を使用していない場合や、不適切なタイプの OREF を使用した場合、InterSystems IRIS はエラーを返します。このエラーは、ObjectScript ターミナルと Python ターミナルで異なります。

ObjectScript Shell

```
TESTNAMESPACE>set x=2
TESTNAMESPACE>set x.Name="Fred Parker"
SET x.Name="Fred Parker"
^
<INVALID OREF>
```

Python Shell

```
>>> x=2
>>> x.Name="Fred Parker"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: 'int' object has no attribute 'Name'
```

このエラーを認識できると便利です。これは、変数が OREF である必要があるのに、OREF でないことを意味します。

- ・ OREF を作成する方法は 1 つのみです。OREF を返すメソッドを使用します。OREF を返すメソッドは、オブジェクト・クラスまたはそれらのサブクラスで定義されます。

以下は、OREF を作成せず、OREF に類似した文字列を作成します。

ObjectScript Shell

```
TESTNAMESPACE>set testthis="4@Sample.Person"
```

Python Shell

```
>>> testthis="<iris.Sample.Person object at 0x000001A1E52FFD20>"
```

- ・ ObjectScript では、変数に OREF が含まれているかどうかをプログラムで判別できます。関数 `$isObject` は、変数に OREF が含まれている場合に 1 (True) を、その他の場合に 0 (False) を返します。

注釈 永続クラスの場合は、`%OpenId()` などのメソッドも OREF を返します。

4.4 ストリーム・インタフェース・クラス

前述のとおり、InterSystems IRIS では、文字列操作の結果を保存するために、一定容量を持つ領域が割り当てられます。割り当てられた容量を超える文字列式が発生すると、`<MAXSTRING>` エラーとなります。上限は 3,641,144 文字です。

この上限を超える長さの値を渡す必要がある場合や、この上限を超える可能性のある値を持つプロパティが必要な場合は、ストリームを使用します。ストリームは、文字列サイズの制限を超える単一の値を含むことができるオブジェクトです。(内部で InterSystems IRIS によって、一時的なグローバルが作成および使用されて、メモリの制限が回避されます。)

InterSystems SQL では、ストリーム・フィールドを使用できますが、いくつかの制限があります。詳細と完全な概要については、“[クラスの定義と使用](#)”を参照してください。また、これらのクラスのインターシステムズ・クラス・リファレンスも参照してください。

注釈 Python で ObjectScript ストリーム・フィールドを使用することはできません。

4.4.1 ストリーム・クラス

InterSystems IRIS の主要なストリーム・クラスでは、`%Stream.Object` クラスによって定義される共通のストリーム・インタフェースを使用します。通常、ストリームは他のオブジェクトのプロパティとして使用し、それらのオブジェクトを保存します。ストリーム・データは、選択するクラスに応じて、外部ファイルまたは InterSystems IRIS グローバルのいずれかに保存できます。

- ・ `%Stream.FileCharacter` クラスおよび `%Stream.FileBinary` クラスは、外部ファイルに書き込まれるストリームに対して使用されます。

(バイナリ・ストリームは、`%Binary` タイプと同種のデータを含み、写真などの大規模なバイナリ・オブジェクトを格納します。文字ストリームは、`%String` タイプと同種のデータを含み、大量のテキストの保存を目的としています。)

- ・ `%Stream.GlobalCharacter` クラスおよび `%Stream.GlobalBinary` クラスは、グローバルに保存されたストリームに対して使用されます。

ストリーム・オブジェクトを操作するには、そのメソッドを使用します。例えば、ストリームにデータを追加するにはこれらのクラスの `Write()` メソッドを使用し、それからデータを読み取るには `Read()` を使用します。ストリーム・インタフェースには、`Rewind()` や `MoveTo()` などの他のメソッドが含まれます。

4.4.2 例

例えば、以下のコードは、グローバル文字ストリームを作成し、データをそれに書き込みます。

ObjectScript

```
Set mystream=##class(%Stream.GlobalCharacter).%New()
Do mystream.Write("here is some text to store in the stream ")
Do mystream.Write("here is some more text")
Write "this stream has this many characters: ",mystream.Size,!
Write "this stream has the following contents: ",!
Write mystream.Read()
```

4.5 コレクション・クラス

一連の関連する値のためのコンテナが必要な場合は、このドキュメントで前述したように、`$LIST` フォーマット・リストおよび多次元配列を使用できます。

クラスを使用したい場合、InterSystems IRIS にはリスト・クラスと配列クラスがあります。これらをコレクションといいます。コレクションの詳細は、“[コレクションを使用した作業](#)” を参照してください。

4.5.1 スタンドアロン・オブジェクトとして使用するリストおよび配列クラス

リスト・オブジェクトを作成するには、以下のクラスを使用します。

- ・ `%Library.ListOfDataTypes` — リテラル値のリストを定義します。
- ・ `%Library.ListOfObjects` — オブジェクト（永続またはシリアル）のリストを定義します。

リスト内の要素は順番に並べられます。リスト内での要素の位置には、1 から N までの範囲の整数のキーを使用してアクセスできます。N は最後の要素の位置です。

リスト・オブジェクトを操作するには、そのメソッドを使用します。以下はその例です。

ObjectScript

```
set Colors = ##class(%Library.ListOfDataTypes).%New()
do Colors.Insert("Red")
do Colors.Insert("Green")
do Colors.Insert("Blue")

write "Number of list items: ", Colors.Count()
write !, "Second list item: ", Colors.GetAt(2)

do Colors.SetAt("Yellow",2)
write !, "New second item: ", Colors.GetAt(2)

write !, "Third item before insertion: ", Colors.GetAt(3)
do Colors.InsertAt("Purple",3)
write !, "Number of items after insertion: ", Colors.Count()
write !, "Third item after insertion: ", Colors.GetAt(3)
write !, "Fourth item after insertion: ", Colors.GetAt(4)

do Colors.RemoveAt(3)
write "Number of items after removing item 3: ", Colors.Count()
```

```
write "List items:"
for i = 1:1:Colors.Count() write Colors.GetAt(i),!
```

Python

```
import iris

Colors=iris.cls("%Library.ListOfDataTypes")._New()
Colors.Insert("Red")
Colors.Insert("Green")
Colors.Insert("Blue")

print("Number of list items:", Colors.Count())
print("Second list item:", Colors.GetAt(2))

Colors.SetAt("Yellow",2)
print("New second item: ", Colors.GetAt(2))

print("Third item before insertion: ", Colors.GetAt(3))
Colors.InsertAt("Purple",3)
print("Number of items after insertion: ", Colors.Count())
print("Third item after insertion: ", Colors.GetAt(3))
print("Fourth item after insertion: ", Colors.GetAt(4))

Colors.RemoveAt(3)
print("Number of items after removing item 3: ", Colors.Count())

print("List items:")
for i in range(1, Colors.Count() + 1) print(Colors.GetAt(i))
```

同様に、配列オブジェクトを作成するには、以下のクラスを使用します。

- ・ **%Library.ArrayOfDataTypes** — リテラル値の配列を定義します。各配列項目には、キーと値があります。
- ・ **%Library.ArrayOfObjects** — オブジェクト (永続またはシリアル) の配列を定義します。各配列項目には、キーとオブジェクト値があります。

配列オブジェクトを操作するには、そのメソッドを使用します。以下はその例です。

ObjectScript

```
set ItemArray = ##class(%Library.ArrayOfDataTypes)._New()
do ItemArray.SetAt("example item","alpha")
do ItemArray.SetAt("another item","beta")
do ItemArray.SetAt("yet another item","gamma")
do ItemArray.SetAt("still another item","omega")
write "Number of items in this array: ", ItemArray.Count()
write !, "Item that has the key gamma: ", ItemArray.GetAt("gamma")
```

Python

```
import iris
ItemArray=iris.cls("%Library.ArrayOfDataTypes")._New()
ItemArray.SetAt("example item", "alpha")
ItemArray.SetAt("another item", "beta")
ItemArray.SetAt("yet another item", "gamma")
ItemArray.SetAt("still another item", "omega")
print("Number of items in this array:", ItemArray.Count())
print("Item that has the key gamma:", ItemArray.GetAt("gamma"))
```

SetAt() メソッドは項目を配列に追加します。最初の引数は追加する要素、2 つ目の引数はキーです。配列要素は、キーの順に並べられます。最初は数値キーで、小さい数値から大きい数値の順に並べられ、次に文字列キーで、アルファベット順に大文字の後に小文字という順に並べられます。例：-2、-1、0、1、2、A、AA、AB、a、aa、ab。

4.5.2 プロパティとしてのリストおよび配列

プロパティをリストまたは配列として定義することもできます。

プロパティをリストとして定義するには、以下の形式を使用します。

Class Member

```
Property MyProperty as list of Classname;
```

Classname がデータ型クラスの場合、InterSystems IRIS は `%Collection.ListOfDT` によって提供されるインタフェースを使用します。Classname がオブジェクト・クラスの場合は、`%Collection.ListOfObj` によって提供されるインタフェースを使用します。

プロパティを配列として定義するには、以下の形式を使用します。

Class Member

```
Property MyProperty as array of Classname;
```

Classname がデータ型クラスの場合、InterSystems IRIS は `%Collection.ArrayOfDT` によって提供されるインタフェースを使用します。Classname がオブジェクト・クラスの場合は、`%Collection.ArrayOfObj` によって提供されるインタフェースを使用します。

4.6 便利な ObjectScript 関数

ObjectScript は、オブジェクト・クラスで使用する以下の関数を提供します。

- ・ `$CLASSMETHOD` では、クラス名およびメソッド名を指定することでクラス・メソッドを実行できます。以下はその例です。

```
TESTNAMESPACE>set class="Sample.Person"
TESTNAMESPACE>set obj=$CLASSMETHOD(class,"%OpenId",1)
TESTNAMESPACE>w obj.Name
Van De Griek,Charlotte M.
```

この関数は、クラス・メソッドを実行する汎用コードを作成する必要があるが、クラス名（またはメソッド名）が事前にわからない場合に便利です。以下はその例です。

ObjectScript

```
//read name of class from imported document
Set class=$list(headerElement,1)
// create header object
Set headerObj=$classmethod(class,"%New")
```

他の関数も同様のシナリオで便利です。

- ・ `$METHOD` では、インスタンスおよびメソッド名を指定することでインスタンス・メソッドを実行できます。以下はその例です。

```
TESTNAMESPACE>set obj=##class(Sample.Person).%OpenId(1)
TESTNAMESPACE>do $METHOD(obj,"PrintPerson")
Name: Van De Griek,Charlotte M.
```

- ・ `$PROPERTY` は、指定されたインスタンスの指定されたプロパティの値を取得または設定します。以下はその例です。

```
TESTNAMESPACE>set obj=##class(Sample.Person).%OpenId(2)
TESTNAMESPACE>write $property(obj,"Name")
Edison,Patrick J.
```

- ・ `$PARAMETER` は、指定されたインスタンスの指定されたクラス・パラメータの値を取得します。以下はその例です。

```
TESTNAMESPACE>set obj=##class(Sample.Person).%OpenId(2)

TESTNAMESPACE>write $parameter(obj,"EXTENTQUERYSPEC")
Name,SSN,Home.City,Home.State
```

- ・ `$CLASSNAME` は、指定されたインスタンスのクラス名を返します。以下はその例です。

```
TESTNAMESPACE>set obj=##class(Sample.Person).%OpenId(1)

TESTNAMESPACE>write $CLASSNAME(obj)
Sample.Person
```

引数がない場合、この関数は現在のコンテキストのクラス名を返します。これは、インスタンス・メソッドで便利です。

4.7 詳細

このページで説明したトピックの詳細は、以下のリソースを参照してください。

- ・ [“クラスの定義と使用”](#) では、InterSystems IRIS でのクラスおよびクラス・メンバの定義方法について説明しています。
- ・ [“クラス定義リファレンス”](#) には、クラス定義で使用するコンパイラ・キーワードのリファレンス情報があります。
- ・ [“インターシステムズ・クラス・リファレンス”](#) には、InterSystems IRIS で提供されるすべての非内部クラスに関する詳細があります。

5

永続オブジェクトと InterSystems IRIS SQL

InterSystems IRIS® の主な特徴は、オブジェクト・テクノロジーと SQL との結合にあります。どのようなシナリオに対しても、最も便利なアクセス・モードを使用できます。このページでは、InterSystems IRIS でこの機能がどのように提供されているのかを説明し、格納されているデータを操作するためのオプションの概要について説明します。

このページで示されている ObjectScript のサンプルは、Samples-Data サンプル (<https://github.com/interSystems/Samples-Data>) のものです。(例えば) **SAMPLES** という名前の専用ネームスペースを作成し、そのネームスペースにサンプルをロードすることをお勧めします。一般的な手順は、“[InterSystems IRIS で使用するサンプルのダウンロード](#)”を参照してください。

5.1 はじめに

InterSystems IRIS は、オブジェクト指向のプログラミング言語が組み込まれたマルチモデル・データ・プラットフォームです。その結果、以下をすべて実行できる柔軟なコードを作成できます。

- ・ SQL を介してデータの一括挿入を実行します。
- ・ オブジェクトを開き、変更し、保存します。したがって、SQL を使用しないで 1 つ以上のテーブルのデータを変更します。
- ・ 新しいオブジェクトを作成、保存し、SQL を使用しないで 1 つ以上のテーブルに行を追加します。
- ・ 多数のオブジェクトに繰り返し処理を実行せずに、SQL を使用して、レコードから、指定された条件に一致する値を取得します。
- ・ オブジェクトを削除し、SQL を使用せずに 1 つ以上のテーブルからレコードを削除します。

つまり、いつでもニーズにあったアクセス・モードを選択できます。

内部では、すべてのアクセスは、直接グローバル・アクセスを介して実行され、必要に応じてユーザもその方法で自身のデータにアクセスできます。(クラス定義がある場合、直接グローバル・アクセスを使用してデータに変更を行うことはお勧めしません。)

5.2 InterSystems SQL

InterSystems IRIS は、InterSystems SQL と呼ばれる SQL の実装を提供します。InterSystems SQL は、メソッド内およびルーチン内で使用できます。

また、InterSystems SQL は、SQL シェル内 (ターミナル) および管理ポータルで直接実行することもできます。これらには、それぞれ、クエリ・プランを表示するためのオプションがあり、クエリをさらに効率的にする方法を特定する際に役立ちます。

InterSystems SQL は、いくつかの例外を除き、すべての初級 SQL-92 標準をサポートし、いくつかの拡張もあります。InterSystems SQL は、インデックス、トリガ、BLOB、およびストアド・プロシージャもサポートしています (これらは典型的な RDBMS 機能であり、SQL-92 標準には含まれません)。完全なリストについては、“[InterSystems SQL の使用法](#)”を参照してください。

5.2.1 ObjectScript からの SQL の使用

以下のいずれかまたは両方の方法を使用して、ObjectScript から SQL を実行できます。

- ・ ダイナミック SQL (%SQL.Statement クラスおよび %SQL.StatementResult クラス)。以下に例を示します。

ObjectScript

```
SET myquery = "SELECT TOP 5 Name, DOB FROM Sample.Person"
SET tStatement = ##class(%SQL.Statement).%New()
SET tStatus = tStatement.%Prepare(myquery)
SET rset = tStatement.%Execute()
DO rset.%Display()
WRITE !,"End of data"
```

ダイナミック SQL は、ObjectScript のメソッドおよびルーチンで使用できます。

- ・ 埋め込み SQL。以下に例を示します。

ObjectScript

```
&sql(SELECT COUNT(*) INTO :myvar FROM Sample.Person)
IF SQLCODE<0 {WRITE "SQLCODE error ",SQLCODE," ",%msg QUIT}
ELSEIF SQLCODE=100 {WRITE "Query returns no results" QUIT}
WRITE myvar
```

埋め込み SQL は、ObjectScript のメソッドおよびルーチンで使用できます。

5.2.2 Python からの SQL の使用

以下のいずれかまたは両方の方法を使用して、Python から SQL を実行できます。

- ・ SQL クエリを直接実行できます。以下に例を示します。

Python

```
import iris
rset = iris.sql.exec("SELECT TOP 5 Name, DOB FROM Sample.Person")
for row in rset:
    print(row)
```

- ・ 最初に SQL クエリを準備してから実行することもできます。以下に例を示します。

Python

```
import iris
statement = iris.sql.prepare("SELECT TOP 5 Name, DOB FROM Sample.Person")
rset = statement.execute()
for row in rset:
    print(row)
```

Python ターミナルまたは Python メソッドでこれらのいずれかの方法を使用して、SQL クエリを実行できます。

5.2.3 SQL へのオブジェクト拡張

オブジェクト・アプリケーションで SQL を使用しやすくするために、InterSystems IRIS には SQL へのオブジェクト拡張が多数あります。

大変興味深い拡張として、“矢印構文”とも呼ばれる暗黙結合演算子 (“->”) を使用して、オブジェクト参照を実行する機能があります。例えば、2 つのクラス、**Contact** と **Region** を参照する **Vendor** クラスがあるとします。暗黙結合演算子を使用すると、関連するクラスのプロパティを参照できます。

SQL

```
SELECT ID, Name, ContactInfo->Name
FROM Vendor
WHERE Vendor->Region->Name = 'Antarctica'
```

また、SQL JOIN 構文を使用しても、同様のクエリ式を記述できます。暗黙結合演算子構文の利点は、簡潔で理解しやすいことです。

5.3 永続クラスに対する特別なオプション

InterSystems IRIS では、すべての永続クラスは **%Library.Persistent** (**%Persistent** と呼ぶ) を拡張します。このクラスは、InterSystems IRIS でのオブジェクト SQL 対応のためのフレームワークの大部分を提供します。永続クラス内では、以下のようなオプションがあります。

- ・ メソッドを使用してオブジェクトを開く、保存する、および削除する機能。

永続オブジェクトを開く場合、永続オブジェクトは、複数のユーザまたは複数のプロセスによって使用される可能性があるため、同時処理ロックの程度を指定します。

オブジェクト・インスタンスを開き、オブジェクト値プロパティを参照すると、システムによってそのオブジェクトも自動的に開かれます。このプロセスをスウィズリングといいます (また、遅延ロードということもあります)。それにより、そのオブジェクトも操作できます。以下の例では、**Sample.Person** オブジェクトを開くと、対応する **Sample.Address** オブジェクトがスウィズルされます。

ObjectScript

```
Set person=##class(Sample.Person).%OpenId(10)
Set person.Name="Andrew Park"
Set person.Address.City="Birmingham"
Do person.%Save()
```

Python

```
import iris
person=iris.cls("Sample.Person")._OpenId(10)
person.Name="Andrew Park"
person.Address.City="Birmingham"
person._Save()
```

同様に、オブジェクトを保存すると、システムによって、そのすべてのオブジェクト値プロパティも自動的に保存されます。これをディープ・セーブといいます。代わりに、シャロウ・セーブを実行するオプションもあります。

- ・ 既定のクエリ (エクステント・クエリ) を使用する機能。既定のクエリは、このクラスのオブジェクトのデータが含まれる SQL 結果セットです。既定では、エクステント・クエリはエクステント内の既存の ID を返します。エクステント・クエリを変更して、返す列を増やすこともできます。

このクラス (または他のクラス) では、追加の [クエリ](#) を定義できます。

- ・ 外部キーとして SQL に投影されるクラス間のリレーションシップを定義する機能。

リレーションシップは、2 つ以上のオブジェクト・インスタンスの相互の関連性を定義する、オブジェクト値プロパティの特別なタイプです。すべてのリレーションシップは 2 つで 1 組みです。リレーションシップのすべての定義には、対応する逆のリレーションシップがあり、それが反対側を定義します。InterSystems IRIS ではデータの参照整合性が自動的に適用され、一方での操作を、他方ですぐに見ることができます。リレーションシップは、メモリ内の振る舞いやディスク上の振る舞いを自動的に管理します。それらは、オブジェクト・コレクションを使用した優れたスケーリングと同時処理も提供します（“[コレクション・クラス](#)” を参照してください）。

- ・ 外部キーを定義する機能。実際は、外部キーを追加して、既存のアプリケーションに参照整合性制約を追加します。新しいアプリケーションの場合、代わりにリレーションシップを定義した方が簡単です。
- ・ これらのクラスにインデックスを定義する機能。

インデックスは、永続クラスのインスタンス全体にわたる検索を最適化するためのメカニズムを提供します。インデックスは、クラスに関連する、要求されること多いデータをソートした特定のサブセットを定義します。これらは、パフォーマンス・クリティカルな検索のオーバーヘッドを削減するのに非常に役立ちます。

インデックスは、そのクラスに属する 1 つまたは複数のプロパティでソートできます。これは、返される結果の順序を制御するのに大変便利です。

またインデックスは、ソートされたプロパティに基づいて、クエリで頻繁に要求される他のデータを格納できます。インデックスの一部として他のデータを含めることによって、インデックスを使用するクエリの性能が大きく向上します。メイン・データ・ストレージにアクセスしなくても、クエリがインデックスを使用して結果セットを生成できます。

- ・ 行が挿入、変更、または削除されるときに何が行われるのかを制御するために、これらのクラスにトリガを定義する機能。
- ・ メソッドおよびクラス・クエリを SQL ストアド・プロシージャとして投影する機能。
- ・ SQL へのプロジェクションを細かく調整する（例えば、SQL クエリのようにテーブルと列名を指定する）機能。
- ・ オブジェクトのデータを格納するグローバルの構造を細かく調整する機能。

注釈 Python では、リレーションシップ、外部キー、およびインデックスを定義することはできません。

5.4 永続クラスの SQL プロジェクション

永続クラスの場合、クラスの各インスタンスは、SQL を使用してクエリおよび操作できる、テーブルの行として使用できます。この例を示すために、このセクションでは管理ポータルとターミナルを使用します。これらについては、このドキュメントで後で説明します。

5.4.1 オブジェクト SQL プロジェクションのデモ

SAMPLES の `Sample.Person` クラスを考えてみます。管理ポータルを使用して、このクラスに対応するテーブルのコンテンツを表示すると、以下に類似したものが表示されます。

Refresh

Close Window

Sample.Person in namespace SAMPLES

#	ID	Age	DOB	FavoriteColors	Name	SSN	Spouse	Home_City	Home_State	Home_Street
1	1	14	03/20/2000	Red	Newton,Dave R.	384-10-6538		Pueblo	AK	6977 First Street
2	2	17	05/30/1997	Green	Waterman,Danielle C.	944-39-5991		Oak Creek	ID	1648 Maple Street
3	3	86	04/01/1928		DeSantis,Christen N.	336-13-6311		Boston	AZ	8572 Maple Street
4	4	55	02/29/1980	Purple	Baker,Marvin Z.	198-22-7709		Queensbury	NV	1243 First Blvd
5	5	80	12/13/1934	Black	Diavolo,Ralph A.	586-13-9662		Hialeah	NY	3880 Maple Pl
6	6	38	10/13/1976		Russell,Paul S.	572-40-8824		Denver	CA	7269 Main Pl
7	7	33	11/26/1981	Purple Purple	Pascal,John X.	468-82-7179		Zanesville	AR	872 Elm Street

(このサンプルはリリースごとに再構築されるため、これは実際に表示されるものと同じデータではありません。)以下の点に注意してください。

- ここに表示される値は、表示値であり、ディスクに格納されている論理値ではありません。
- 最初の列 (#) は、表示されているこのページ内の行番号です。
- 2 番目の列 ([ID]) は、このテーブルの行に対する一意の識別子であり、このクラスのオブジェクトを開くときは、この識別子を使用します (このクラスではこれらの識別子は整数ですが、整数ではないこともあります)。

このテーブルは、**SAMPLES** データベースが構築されるたびに新しく生成されるため、この場合、これらの番号は同一になっています。実際のアプリケーションでは、いくつかのレコードが削除されている可能性があるため、[ID] の値には相違があり、それらの値は行番号と一致しません。

ターミナルで、一連のコマンドを使用して、最初の人を調べることができます。

ObjectScript Shell

```
SAMPLES>set person=##class(Sample.Person).%OpenId(1)

SAMPLES>write person.Name
Newton,Dave R.
SAMPLES>write person.FavoriteColors.Count()
1
SAMPLES>write person.FavoriteColors.GetAt(1)
Red
SAMPLES>write person.SSN
384-10-6538
```

Python Shell

```
>>> person=iris.cls("Sample.Person")._OpenId(1)
>>> print(person.Name)
Newton,Dave R.
>>> print(person.FavoriteColors.Count())
1
>>> print(person.FavoriteColors.GetAt(1))
Red
>>> print(person.SSN)
384-10-6538
```

これらは、SQL を使用した場合と同じ値です。

5.4.2 オブジェクト SQL プロジェクションの基本事項

継承はリレーショナル・モデルの一部ではないので、クラス・コンパイラは、永続クラスを“平坦化”したものをリレーショナル・テーブルとして投影します。以下の表は、さまざまなオブジェクト要素のいくつかが SQL にどのように投影されるのかを示しています。

オブジェクト・コンセプト	SQL コンセプト
パッケージ	スキーマ
クラス	テーブル
プロパティ	フィールド
埋め込みオブジェクト	一連のフィールド
リスト・プロパティ	リスト・フィールド
配列プロパティ	子テーブル
ストリーム・プロパティ	BLOB または CLOB
インデックス	インデックス
ストアド・プロシージャとしてマークされるクラス・メソッド	ストアド・プロシージャ

投影されたテーブルには、継承されたフィールドを含む、そのクラスに適切なすべてのフィールドが含まれます。

5.4.3 クラスとエクステント

InterSystems IRIS は、従来にない強力な解釈によるオブジェクトテーブル・マッピングを使用します。

永続クラスのすべての格納されているインスタンスによって、クラスのエクステントと呼ばれるものが構成され、インスタンスは、それがインスタンスとなっているクラスそれぞれのエクステントに属します。したがって、以下のようになります。

- ・ 永続クラス **Person** にサブクラス **Student** がある場合、**Person** エクステントには、**Person** のすべてのインスタンスと **Student** のすべてのインスタンスが含まれます。
- ・ クラス **Student** のどのインスタンスの場合も、そのインスタンスは **Person** エクステントおよび **Student** エクステントに含まれます。

インデックスは、インデックスが定義されるクラスの全範囲を自動的にカバーします。**Person** に定義されているインデックスには、**Person** インスタンスと **Student** インスタンスの両方が含まれます。**Student** エクステントに定義されているインデックスには、**Student** インスタンスだけが含まれます。

サブクラスは、そのスーパークラスで定義されていない追加のプロパティも定義できます。これらは、サブクラスのエクステントでは使用できますが、スーパークラスのエクステントでは使用できません。例えば、**Student** エクステントには **FacultyAdvisor** フィールドが含まれますが、これは **Person** エクステントには含まれません。

前述の点は、InterSystems IRIS では、同じタイプのレコードをすべて取得するクエリを比較的簡単に作成できることを意味します。例えば、すべてのタイプの人々をカウントする場合、**Person** テーブルに対してクエリを実行できます。学生のみをカウントする場合、同じクエリを **Student** テーブルに対して実行します。これとは対照的に、他のオブジェクト・データベースでは、すべてのタイプの人々をカウントするには、テーブルを結合する複雑なクエリを作成する必要があり、別のサブクラスが追加されるたびにこのクエリを更新する必要があります。

5.5 オブジェクト ID

各オブジェクトは、それが属する各エクステント内で一意の ID を持っています。多くの場合、この ID を使用してそのオブジェクトを操作します。この ID は、**%Persistent** クラスの、以下のよく使用されるメソッドの引数です。

- ・ `%DeleteId()`
- ・ `%ExistsId()`
- ・ `%OpenId()`

このクラスには、ID を使用する他のメソッドもあります。

5.5.1 ID はどのように決まるか

最初にオブジェクトを保存するときに、InterSystems IRIS によって ID 値が割り当てられます。この割り当ては永続的であり、オブジェクトの ID は変更できません。他のオブジェクトが削除されたり、変更された場合でも、オブジェクトに新しい ID が割り当てられることはありません。

どの ID も、そのエクステント内で一意です。

オブジェクトの ID は、以下のように決定されます。

- ・ 大部分のクラスでは、ID は、既定ではそのクラスのオブジェクトが保存されるときに順番に割り当てられる整数です。
- ・ 親子リレーションシップの子として使用されるクラスでは、ID は以下のようにフォーマットされます。

```
parentID || childID
```

parentID は親オブジェクトの ID であり、childID は、子オブジェクトが親子リレーションシップの子として使用されていない場合に受け取る ID です。例：

```
104 || 3
```

この ID は 3 番目に保存された子であり、その親はそれ自体のエクステントに ID 104 を持っています。

- ・ このクラスが、タイプ `IdKey` のインデックスを持っていて、そのインデックスが特定のプロパティに対するものである場合、そのプロパティ値が ID として使用されます。

```
SKU-447
```

また、そのプロパティ値も変更できません。

- ・ このクラスが、タイプ `IdKey` のインデックスを持っていて、そのインデックスが複数のプロパティに対するものである場合、それらのプロパティ値が連結されて ID を形成します。以下に例を示します。

```
CATEGORY12 || SUBCATEGORYA
```

また、これらのプロパティ値も変更できません。

5.5.2 ID へのアクセス

オブジェクトの ID 値にアクセスするには、そのオブジェクトが `%Persistent` から継承する `%Id()` インスタンス・メソッドを使用します。

ObjectScript Shell

```
SAMPLES>set person=##class(Sample.Person).%OpenId(2)
```

```
SAMPLES>write person.%Id()  
2
```


Python Shell

```
>>> person = iris.cls("Sample.Person")._OpenId(2)
>>> print(person._Id())
2
```

SQL では、オブジェクトの ID 値は、[%Id] という擬似フィールドとして使用できます。管理ポータルでテーブルを参照する場合には、[%Id] 擬似フィールドは ID というキャプションで表示されます。

Refresh

Close Window

Sample.Person in namespace SAMPLES

#	ID	Age	DOB	FavoriteColors	Name	SSN	Spouse	Home_City	Home_State	Home_Street
1	1	14	03/20/2000	Red	Newton,Dave R.	384-10-8538		Pueblo	AK	6977 First Street
2	2	17	05/30/1997	Green	Waterman,Danielle C.	944-39-5991		Oak Creek	ID	1648 Maple Street
3	3	86	04/01/1928		DeSantis,Christen N.	336-13-6311		Boston	AZ	8572 Maple Street
4	4	55	02/29/1980	Purple	Baker,Marvin Z.	198-22-7709		Queensbury	NV	1243 First Blvd
5	5	80	12/13/1934	Black	Diavolo,Ralph A.	586-13-9662		Hialeah	NY	3880 Maple Plaza
6	6	38	10/13/1976		Russell,Paul S.	572-40-8824		Denver	CA	7269 Main Plaza
7	7	33	11/26/1981	Purple Purple	Pascal,John X.	468-82-7179		Zanesville	AR	872 Elm Street

キャプションはこのようになっていても、擬似フィールドの名前は [%Id] です。

5.6 ストレージ

各永続クラス定義には、そのクラス・プロパティが、それが実際に格納されているグローバルにどのようにマップされるかを示す情報が含まれています。この情報は、クラス・コンパイラによってクラスに対して生成され、開発者が変更してリコンパイルしたときに更新されます。

5.6.1 ストレージ定義の確認

この情報を確認すると役立つ場合があります、まれに、その詳細のいくつかを(注意深く)変更することが必要になります。永続クラスの場合、[統合開発環境 \(IDE\)](#) はクラス定義の一部として、次に示すような定義を表示します。

```
<Storage name="Default">
<Data name="PersonDefaultData"><Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>Name</Value>
</Value>
<Value name="3">
<Value>SSN</Value>
</Value>
<Value name="4">
<Value>DOB</Value>
</Value>
...
</Storage>
```


5.6.2 永続クラスによって使用されるグローバル

ストレージ定義には、データが格納されるグローバルを指定するいくつかの要素が含まれています。

```
<DataLocation>^Sample.PersonD</DataLocation>
<IdLocation>^Sample.PersonD</IdLocation>
<IndexLocation>^Sample.PersonI</IndexLocation>
...
<StreamLocation>^Sample.PersonS</StreamLocation>
```

既定では、既定のストレージで以下ようになります。

- ・ クラス・データは、そのクラスのデータ・グローバルに格納されます。その名前は、完全なクラス名（パッケージ名を含む）で始まります。その名前に“D”が追加されます。例えば、`Sample.PersonD` のようになります。
- ・ インデックス・データは、そのクラスのインデックス・グローバルに格納されます。その名前は、クラス名で始まり、“I”で終わります。例えば、`Sample.PersonI` のようになります。
- ・ 保存されるストリーム・プロパティは、そのクラスのストリーム・グローバルに格納されます。その名前は、クラス名で始まり、“S”で終わります。例えば、`Sample.PersonS` のようになります。

重要 完全なクラス名が長い場合、システムでは、代わりにクラス名のハッシュした形式が自動的に使用されます。したがって、ストレージ定義を表示したときに、`^package1.pC347.VeryLongCla4F4AD` のようなグローバル名が表示されることがあります。何らかの理由で、クラスのデータ・グローバルを使用して直接作業する場合は、ストレージ定義を調べて、グローバルの実際の名前を確認する必要があります。

グローバル名がどのように決まるかは、“クラスの定義と使用”の“[グローバル](#)”を参照してください。

5.6.3 格納されるオブジェクトの既定の構造

一般的なクラスの場合、大部分のデータはデータ・グローバルに格納され、それには以下のノードが含まれています。

ノード	ノードのコンテンツ
<code>^full_class_nameD(id)</code> <code>full_class_name</code> は、パッケージを含む完全なクラス名であり、31 文字までの長さにする必要がある場合はハッシュされます。また、 <code>id</code> はオブジェクト ID であり、これについては、“ オブジェクト ID ”で説明しています。	<code>\$ListBuild</code> によって返されるフォーマットのリスト。 このリストでは、格納されるプロパティは、ストレージ定義の <code><Value></code> 要素の <code>name</code> 属性によって指定される順序でリストされます。 当然ながら、一時プロパティは格納されません。ストリーム・プロパティは、そのクラスのストリーム・グローバルに格納されます。

使用例は、“[格納されているデータの確認](#)”を参照してください。

5.6.4 メモ

以下の点に注意してください。

- ・ 格納されたデータを持つクラスのストレージを再定義したり、削除しないでください。その場合、ストレージを手動で再作成しなければならなくなります。それは、次にそのクラスをコンパイルしたときに作成される新しい既定のストレージは、そのクラスに必要なストレージと合わない可能性があるためです。
- ・ 開発中に、クラスのストレージ定義のリセットが必要になる場合があります。それは、データも削除して、後でそれを再ロードまたは再生成する場合に実行できます。

- ・ 既定では、開発中にプロパティを追加および削除すると、システムによってスキーマ進化というプロセスが使用されて自動的にストレージ定義が更新されます。

例外は、<Type> 要素に対して既定以外のストレージ・クラスを使用する場合です。既定は `%Storage.Persistent` です。このストレージ・クラスを使用しない場合、InterSystems IRIS によってストレージ定義が更新されることはありません。

5.7 永続クラスおよびテーブルを作成するためのオプション

永続クラスとそれに対応する SQL テーブルを作成するには、次のいずれかを実行します。

- ・ IDE を使用して、`%Persistent` に基づき、クラスを定義します。そのクラスをコンパイルすると、システムによってテーブルが作成されます。
- ・ 管理ポータルで、データ移行ウィザードを使用できます。これにより、外部テーブルが読み取られ、いくつかの詳細設定を求めるプロンプトが表示され、`%Persistent` をベースにしたクラスが作成され、対応する SQL テーブルにレコードがロードされます。

後で、ウィザードを再び実行することで、クラスを再定義しないで追加のレコードをロードできます。

- ・ 管理ポータルで、リンク・テーブル・ウィザードを使用できます。これにより、外部テーブルが読み取られ、いくつかの詳細設定を求めるプロンプトが表示され、外部テーブルにリンクされているクラスが生成されます。クラスは、実行時にデータを外部テーブルから取得します。

これは、特別な場合であり、このドキュメントでは詳しく説明しません。

- ・ InterSystems SQL で、CREATE TABLE または他の DDL 文を使用します。この方法でもクラスが作成されます。
- ・ ターミナル（またはコード）で、`%SQL.Util.Procedures` の `CSVTOCLASS()` メソッドを使用します。詳細は、`%SQL.Util.Procedures` のクラスリファレンスを参照してください。

5.8 データへのアクセス

永続クラスに関連付けられたデータにアクセス、そのデータを変更および削除するには、以下のいずれかまたはすべてを実行するコードを作成できます。

- ・ 永続クラスのインスタンスを開き、それらを変更および保存します。
- ・ 永続クラスのインスタンスを削除します。
- ・ 埋め込み SQL を使用します。
- ・ ダイナミック SQL (SQL 文と結果セット・インタフェース) を使用します。
- ・ Python から SQL を使用します。
- ・ 直接グローバル・アクセスのための低レベルのコマンドおよび関数を使用します。この手法は、格納されている値を取得する場合以外は、お勧めしません。それは、この手法では、オブジェクトおよび SQL インタフェースによって定義されているロジックがバイパスされるためです。

InterSystems SQL は、以下のような場合に適しています。

- ・ 最初の時点では開くインスタンスの ID が不明であり、代わりに、入力条件に基づいて 1 つ以上のインスタンスを選択する場合。

- ・ 一括ロードまたは一括変更を実行する場合。
- ・ オブジェクト・インスタンスを開かずにそのデータを表示する場合
(ただし、オブジェクト・アクセスを使用する場合、同時処理ロックの程度を制御できます。データを変更する予定がない場合は、最小限の同時処理ロックを使用できます)。
- ・ SQL に精通している場合。

オブジェクト・アクセスは、以下のような場合に適しています。

- ・ 新しいオブジェクトを作成する場合。
- ・ 開くインスタンスの ID がわかっている場合。
- ・ SQL を使用するよりも、プロパティの値を設定する方が直感的に理解しやすい場合。

5.9 格納されているデータの確認

このセクションでは、任意の永続オブジェクトに対して、オブジェクト・アクセス、SQL アクセス、および直接グローバル・アクセスを使用して同じ値が見えることを例示します。

IDE では、**Sample.Person** クラスを表示すると、次のプロパティ定義が表示されます。

```
/// Person's name.
Property Name As %String(POPSPEC = "Name()") [ Required ];

...

/// Person's age.<br>
/// This is a calculated field whose value is derived from <property>DOB</property>.
Property Age As %Integer [ details removed for this example ];

/// Person's Date of Birth.
Property DOB As %Date(POPSPEC = "Date()");
```

ターミナルでは、格納されているオブジェクトを開き、そのプロパティ値を書き込みます。

ObjectScript Shell

```
SAMPLES>set person=##class(Sample.Person)._OpenId(1)

SAMPLES>w person.Name
Newton,Dave R.
SAMPLES>w person.Age
21
SAMPLES>w person.DOB
58153
```

Python Shell

```
>>> person=iris.cls("Sample.Person")._OpenId(1)
>>> print(person.Name)
Newton, Dave R.
>>> print(person.Age)
21
>>> print(person.DOB)
58153
```

ここでは、DOB プロパティのリテラルの格納されている値 (リテラル) が表示されます。代わりに、このプロパティの表示値を返すメソッドを呼び出すこともできます。

ObjectScript Shell

```
SAMPLES>write person.DOBLogicalToDisplay(person.DOB)
03/20/2000
```

Python Shell

```
>>> print(iris.cls("%Date").LogicalToDisplay(person.DOB))
03/20/2000
```

管理ポータルでは、このクラスの格納されているデータを参照できます。以下のように表示されます。

Refresh

Close Window

Sample.Person in namespace SAMPLES

#	ID	Age	DOB	FavoriteColors	Name	SSN	Spouse	Home_City	Home_State	Home_Street
1	1	14	03/20/2000	Red	Newton,Dave R.	384-10-6538		Pueblo	AK	6977 First Street
2	2	17	05/30/1997	Green	Waterman,Danielle C.	944-39-5991		Oak Creek	ID	1648 Maple Street
3	3	86	04/01/1928		DeSantis,Christen N.	336-13-6311		Boston	AZ	8572 Maple Street
4	4	55	02/29/1980	Purple	Baker,Marvin Z.	198-22-7709		Queensbury	NV	1243 First Blvd
5	5	80	12/13/1934	Black	Diavolo,Ralph A.	586-13-9662		Hialeah	NY	3880 Maple Pl
6	6	38	10/13/1976		Russell,Paul S.	572-40-8824		Denver	CA	7269 Main Pl
7	7	33	11/26/1981	Purple Purple	Pascal,John X.	468-82-7179		Zanesville	AR	872 Elm Street

この場合、DOB プロパティの表示値を表示できます。(ポータルでは、クエリを実行するもう 1 つのオプションがあり、そのオプションを使用すると、その結果に対して論理モードと表示モードのどちらを使用するのか制御できます。)

ポータルでは、このクラスのデータが格納されているグローバルを参照することもできます。

Global Search Mask: ^Sample.PersonD
Display
Cancel

Search History: ^Sample.PersonD
Maximum Rows: 100
Allow Edit

1: ^Sample.PersonD = 200
2: ^Sample.PersonD(1) = \$lb("", "Newton,Dave R.", "384-10-6538", 58153, \$lb("6977 First Street", "Pueblo", "AK", 63163), \$lb("9984 Second Blvd", "Washington", "MN", 42829), "", \$lb("Red"))
3: ^Sample.PersonD(2) = \$lb("", "Waterman,Danielle C.", "944-39-5991", 57128, \$lb("1648 Maple Street", "Oak Creek", "ID", 63163), \$lb("9984 Second Blvd", "Washington", "MN", 42829), "", \$lb("Red"))
4: ^Sample.PersonD(3) = \$lb("", "DeSantis,Christen N.", "336-13-6311", 31867, \$lb("8572 Maple Street", "Boston", "AZ", 63163), \$lb("9984 Second Blvd", "Washington", "MN", 42829), "", \$lb("Red"))
5: ^Sample.PersonD(4) = \$lb("", "Baker,Marvin Z.", "198-22-7709", 43523, \$lb("1243 First Blvd", "Queensbury", "NV", 63163), \$lb("9984 Second Blvd", "Washington", "MN", 42829), "", \$lb("Red"))

また、ターミナルでは、このインスタンスを含むグローバル・ノードの値を ObjectScript で書き込むことができます。

```
zwrite ^Sample.PersonD("1")
^Sample.PersonD(1)=$lb("", "Newton,Dave R.", "384-10-6538", 58153, $lb("6977 First
Street", "Pueblo", "AK", 63163),
$lb("9984 Second Blvd", "Washington", "MN", 42829), "", $lb("Red"))
```

スペースの関係で、最後の例には追加の改行が含まれています。

5.10 InterSystems SQL に対して生成されたコードのストレージ

InterSystems SQL の場合、データにアクセスするための再利用可能なコードがシステムによって生成されます。

最初に SQL 文を実行するときに、InterSystems IRIS によってクエリが最適化され、データを取得するコードが生成され、格納されます。コードは、最適化されたクエリ・テキストと共にクエリ・キャッシュに格納されます。このキャッシュは、コードのキャッシュであり、データのキャッシュではありません。

後で SQL 文を実行すると、InterSystems IRIS によって最適化され、そのクエリのテキストとクエリ・キャッシュの項目が比較されます。InterSystems IRIS によって、格納されているクエリが、指定されたクエリと一致していると判断された場合 (空白などのわずかな相違を除く)、そのクエリに対して格納されているコードが使用されます。

クエリ・キャッシュを表示し、その中に含まれている任意の項目を削除できます。

5.11 詳細

このページで説明したトピックの詳細は、以下のリソースを参照してください。

- ・ [“クラスの定義と使用”](#) では、InterSystems IRIS でのクラスおよびクラス・メンバの定義方法について説明しています。
- ・ [“クラス定義リファレンス”](#) には、クラス定義で使用するコンパイラ・キーワードのリファレンス情報があります。
- ・ [“InterSystems SQL の使用法”](#) では、InterSystems SQL の使用方法、およびこの言語を使用できる場所について説明しています。
- ・ [“InterSystems SQL リファレンス”](#) には、InterSystems SQL のリファレンス情報があります。
- ・ [“グローバルの使用法”](#) には、InterSystems IRIS でグローバルに永続オブジェクトを格納する方法が記載されています。
- ・ [“インターシステムズ・クラス・リファレンス”](#) には、InterSystems IRIS で提供されるすべての非内部クラスに関する情報があります。

6

ネームスペースとデータベース

このページでは、InterSystems IRIS® でデータとコードがどのように編成されるのかを説明します。

6.1 ネームスペースとデータベースの概要

InterSystems IRIS では、どのコードも、論理エンティティであるネームスペースで実行されます。ネームスペースはデータとコードへのアクセスを提供し、それらは (通常) 複数のデータベースに格納されています。データベースは、ファイル (IRIS.DAT ファイル) です。InterSystems IRIS は、ユーザが使用するための一連のネームスペースとデータベースを提供し、ユーザは追加のネームスペースとデータベースを定義できます。

ネームスペースでは、以下のオプションを使用できます。

- ・ ネームスペースには、コードを格納するための既定のデータベースがあります。これが、そのネームスペースのルーチン・データベースです。

ネームスペースでコードを作成する場合、他の考慮事項がない限り、コードはそのルーチン・データベースに格納されます。同様に、ユーザがコードを呼び出す場合、他の考慮事項がない限り、InterSystems IRIS によってそのデータベースでコードが検索されます。

- ・ ネームスペースには、作成した永続クラスとグローバルのデータを格納するための既定のデータベースもあります。これが、そのネームスペースのグローバル・データベースです。

したがって、例えば、ユーザがデータに (任意の方法で) アクセスする場合、他の考慮事項がない限り、InterSystems IRIS によってそのデータベースからそれが取得されます。

グローバル・データベースはルーチン・データベースと同じデータベースでもかまいませんが、多くの場合、別のデータベースにした方が管理の面で便利です。

- ・ ネームスペースには、一時的なストレージ用の既定のデータベースがあります。
- ・ ネームスペースには、他のデータベースに格納されている追加のデータとコードへのアクセスを提供するマッピングを含めることができます。具体的には、既定以外のデータベースに格納されているルーチン、クラス・パッケージ、グローバル全体、および特定のグローバル・ノードを参照するマッピングを定義できます。(これらの種類のマッピングは、それぞれ、ルーチン・マッピング、パッケージ・マッピング、グローバル・マッピングおよび添え字レベル・マッピングと呼ばれます。)

マッピングを使用してデータベースへのアクセスを提供する場合は、そのデータベースの一部のみに対するアクセスを提供します。ネームスペースから、そのデータベースのマッピングされていない部分へは、読み取り専用方式でもアクセスできません。

また、マッピングを定義するときは、それがネームスペースの構成にのみ作用することを理解してください。それによって、コードやデータの現在の場所が変わることはありません。したがって、マッピングを定義するときは、コードや

データ (存在する場合) を、その現在の場所から、ネームスペースによって予期される場所に移動することも必要です。

マッピングの定義は、データベース管理タスクです。クラスやテーブル定義、あるいはアプリケーション・ロジックを変更する必要はありません。

- ・ 作成するネームスペースはどれでも、InterSystems IRIS コード・ライブラリの大部分にアクセスできます。このコードを使用できるのは、InterSystems IRIS によって、ユーザが作成するネームスペース専用のマッピングが自動的に確立されるためです。

このドキュメントでは、これらのクラスのいくつかについて述べました。特定の用途のツールを見つけるには、“インターシステムズ・プログラミング・ツールの索引” を使用してください。

- ・ ネームスペースを定義する場合、相互運用対応になるように定義できます。つまり、対象のネームスペースにプロダクションを定義できるということです。プロダクションとは、InterSystems IRIS の相互運用機能を使用して複数の個別ソフトウェア・システムを統合するプログラムです。詳細は、“相互運用プロダクションの概要” を参照してください。

マッピングは、データとコードを共有するための便利で強力な手段を提供します。どのデータベースも、複数のネームスペースから使用できます。例えば、このページで後述する、すべての顧客ネームスペースからアクセスできるいくつかのシステム・データベースがあります。

ネームスペースの構成は、定義した後に変更でき、InterSystems IRIS は、1 つのデータベースから別のデータベースにコードとデータを移動するためのツールを備えています。したがって、必要であれば、開発中にコードとデータを再編成することができます。これにより、(スケーリングなど) InterSystems IRIS アプリケーションの再構成が簡単に実行できます。

6.1.1 ロック、グローバル、およびネームスペース

グローバルには複数のネームスペースからアクセスできるため、InterSystems IRIS は、グローバルの**ロック・メカニズム**用に、自動クロスネームスペースのサポートを提供しています。特定のグローバルに対するロックは、そのグローバルを格納しているデータベースを使用するすべてのネームスペースに自動的に適用されます。

6.2 データベースの基本事項

InterSystems IRIS データベースは **IRIS.DAT** ファイルです。データベースは、管理ポータルを使用して作成します。また、既存の InterSystems IRIS データベースがある場合は、それが認識されるように InterSystems IRIS を構成できます。

6.2.1 データベース構成

どのデータベースに対しても、InterSystems IRIS では、以下の構成詳細が必要です。

- ・ データベースの論理名。
- ・ **IRIS.DAT** ファイルが配置されているディレクトリ。管理ポータルでデータベースを作成するときに、システム管理者用ディレクトリ (install-dir/**Mgr**) 内のサブディレクトリを選択するか作成するように求められますが、データベース・ファイルは、便利なディレクトリであればどこにでも格納できます。

Tip ヒン 論理名と、**IRIS.DAT** ファイルが格納されるディレクトリに同じ文字列を使用すると便利です。システム提供のト InterSystems IRIS データベースは、この規則に従っています。

追加のオプションは以下のとおりです。

- ・ このデータベースによって使用されるファイル・ストリームに使用するための既定のディレクトリ。

このディレクトリへの書き込みアクセスがユーザに必要なため、これは重要です。書き込みアクセスできない場合、コードでファイル・ストリームを作成できなくなります。

- ・ 新しいグローバルの照合。
- ・ 初期サイズと他の物理的特徴。
- ・ ジャーナリングを有効または無効にするオプション。ジャーナリングは、クラッシュ後の最新状態リカバリやシステム・リカバリ中のデータのリストアのために、InterSystems IRIS データベースに行われた変更を記録します。

多くの場合、ジャーナリングを有効にすると役立ちます。ただし、指定された一時作業領域に対しては、ジャーナリングを無効にする場合があります。例えば、**IRISTEMP** データベースはジャーナリングしません。

- ・ 読み取り専用として使用するためにこのデータベースをマウントするオプション。
読み取り専用データベースにグローバルを設定しようとすると、InterSystems IRIS によって <PROTECT> エラーが返されます。

多くの場合、システムが実行している間に、データベースの属性を作成、消去、または修正できます。

6.2.2 データベースの特徴

各データベースで、InterSystems IRIS は、実際のデータとデータ編成メタデータの両方に対して物理的整合性を保証します。この整合性は、データベースへの書き込み中にエラーが発生した場合でも保証されます。

データベースは、手動で操作しなくても、必要に応じて自動的に拡張されます。特定のデータベースのサイズの増大が予想され、どれくらい大きくなるのか判別できる場合は、その初期サイズを、予想される最終的なサイズに近い値に設定することで“事前に拡張”できます。そのようにすると、パフォーマンスが向上します。

InterSystems IRIS には、高可用性と復元可能性を実現するいくつかの方法があります。これには、以下のものがあります。

- ・ ジャーナリング – 前述のとおりです。
- ・ ミラーリング – 2 つの InterSystems IRIS システム間における、迅速で、信頼性の高い、強固な自動フェイルオーバーを提供し、ミラーリングは企業にとって理想的な自動フェイルオーバー高可用性ソリューションとなります。
- ・ クラスタリング – クラスタリングを提供するオペレーティング・システムに対して、その完全なサポートを提供します。

InterSystems IRIS は、複数のシステム間で、データ、アプリケーション・ロジック、および処理を分散するテクノロジーを備えています。これは、ECP(エンタープライズ・キャッシュ・プロトコル)と呼ばれます。マルチサーバ・システムでは、InterSystems IRIS データベース・サーバのネットワークは共有リソースとして構成でき、それらの間でデータをシームレスに分散して、データ・ストレージとアプリケーションの処理を共有できます。これにより、高い拡張性と、自動フェイルオーバーおよびリカバリが実現されます。

6.2.3 データベースの移植性

InterSystems IRIS データベースは、複数のプラットフォーム間およびバージョン間で移植可能ですが、以下の注意事項があります。

- ・ さまざまなプラットフォームで、ファイルはどれもビッグ・エンディアン (最上位のバイトが先頭) とリトル・エンディアン (最下位のバイトが先頭) のいずれかです。

InterSystems IRIS には、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 `cvendian` があります。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。

6.3 システム提供データベース

InterSystems IRIS には、以下のデータベースが用意されています。

- ・ **IRISLIB** – これは、読み取り専用データベースであり、オブジェクト、データ型、ストリーム、コレクション・クラス、および多数の他のクラス定義が含まれます。また、システム・インクルード・ファイル、生成された INT コード (大部分のクラスの場合)、および生成された OBJ コードも含まれます。
- ・ **ENSLIB** – これは、読み取り専用データベースであり、InterSystems IRIS の相互運用機能に必要な追加コードが含まれます。こういった機能とは、具体的には、個別のソフトウェア・システムを統合するプロダクションを作成する機能です。

作成したネームスペースが相互運用対応の場合、そのネームスペースからこのデータベース内のコードにアクセスできます。

- ・ **IRISSYS** (システム管理者用データベース) – このデータベースには、システム管理に関連するユーティリティおよびデータが含まれています。これは、ユーザ固有のカスタム・コードおよびデータを格納することと、更新時にそのコードおよびデータを保持することを目的としています。

このデータベースには、以下が含まれます (含めることができます)。

- ユーザやロールなどのセキュリティ要素 (事前定義項目とユーザが追加する項目の両方)。
セキュリティ上の理由により、管理ポータルでは、このデータは他のデータとは異なる扱いになります。例えば、ユーザとそのパスワードのテーブルは表示できません。
- NLS (各国言語サポート) クラスが使用するデータ (数の形式、文字のソート順序、その他の詳細)。追加のデータをロードできます。
- 独自のコードおよびデータ。更新時にこれらの項目が保持されるようにするには、[IRISSYS データベースおよびカスタム項目](#)の名前付け規約を使用します。

注意 **IRISSYS** データベースの移行、置き換え、または削除はサポートされていません。

このデータベースが配置されるディレクトリは、システム管理者用ディレクトリです。メッセージ・ログ (`messages.log`) は、他のログ・ファイルと同様に、このディレクトリに書き込まれます。

- ・ **IRISAUDIT** – イベントの記録を有効にすると、InterSystems IRIS によって監査データがこのデータベースに書き込まれます。
- ・ **IRISTEMP** – これは、InterSystems IRIS によって一時ストレージ用に使用され、ユーザも同じ用途に使用できます。具体的には、このデータベースには一時グローバルが含まれています。詳細は、“グローバルの使用法” の “[一時グローバルと IRISTEMP データベース](#)” を参照してください。
- ・ **IRISLOCALDATA** – InterSystems IRIS によって内部で使用するアイテム (キャッシュされた SQL クエリや CSP セッション情報など) が含まれます。

注釈 顧客アプリケーションが **IRISLOCALDATA** データベースと直接やり取りすることのないようにしてください。このデータベースは、純粋に InterSystems IRIS での内部使用を目的としています。

IRISSYS のその他の詳細は、“[リソースの使用による資源の保護](#)” を参照してください。

6.4 %SYS ネームスペース

%SYS ネームスペースは、一部のネームスペースでのみ使用可能なコード (セキュリティ要素、サーバ構成などを操作するコード) へのアクセスを提供します。

このネームスペースの場合、既定のルーチン・データベースおよび既定のグローバル・データベースは、IRISSYS です。特定の名前付け規約に従う場合、独自のコードとグローバルをこのネームスペースに作成し、それを [IRISSYS データベース](#) に格納できます。

6.5 IRISSYS データベースおよびカスタム項目

IRISSYS データベースに項目を作成できます。InterSystems IRIS アップグレードをインストールすると、このデータベースがアップグレードされます。このアップグレード中に、カスタム項目の名前付け規約に従っていないと、いくつかの項目が削除されます。

項目が上書きされないようにこのデータベースにコードまたはデータを追加するには、以下のいずれかを実行します。

- ・ %SYS ネームスペースに移動し、項目を作成します。このネームスペースの場合、既定のルーチン・データベースおよび既定のグローバル・データベースは、どちらも IRISSYS です。以下の名前付け規約を使用して、項目がアップグレード・インストールの影響を受けないようにします。
 - － クラス：パッケージを Z または z で始めます。
 - － ルーチン：名前を Z、z、%Z、または %z で始めます。
 - － グローバル：名前を ^Z、^z、^%Z、または ^%z で始めます。
- ・ どのネームスペースでも、以下の名前で作成します。
 - － ルーチン：名前を %Z または %z で始めます。
 - － グローバル：名前を ^%Z または ^%z で始めます。

注釈 %ZEN* や %ZHLIB* など、いくつかの %Z ルーチン名の使用はインターシステムズによって予約済みです。

ネームスペースの標準マッピングにより、これらの項目は IRISSYS に書き込まれます。

MAC コードとインクルード・ファイルはアップグレードによる影響を受けません。

6.6 ネームスペースで何にアクセス可能か

ネームスペースを作成すると、システムによってそのネームスペースに対するマッピングが自動的に定義されます。その結果、そのネームスペース内で以下の項目を使用できるようになります (これらの項目に対する適切な許可を所持するユーザとしてログインした場合)。

- ・ パッケージ名がパーセント記号 (%) で始まるクラス。これには、InterSystems IRIS で提供されるクラスの大部分が含まれますが、すべてではありません。
- ・ このネームスペース用のルーチン・データベースに格納されているすべてのコード。

- ・ このネームスペース用のグローバル・データベースに格納されているすべてのデータ。
- ・ 名前がパーセント記号で始まるルーチン。
- ・ 名前がパーセント記号で始まるインクルード・ファイル。
- ・ 名前がキャラクタおよびパーセント記号 (%) で始まるグローバル。このようなグローバルは、一般的にパーセント・グローバルと呼ばれます。グローバル・マッピングまたは添え字レベル・マッピングを使用すれば、パーセント・グローバルを格納する場所を変更できますが、これらの表示には影響しません。パーセント・グローバルはすべてのネームスペースで常に表示されます。
- ・ `^IRIS.TempUser` で始まる名前を持つユーザ自身のグローバル。例えば、`^IRIS.TempUser.MyApp`。そのようなグローバルを作成すると、それらのグローバルは **IRISTEMP** データベースに書き込まれます。
- ・ ネームスペースが相互運用対応の場合、**Ens** パッケージおよび **EnsLib** パッケージでコードを使用できます。**CSPX** パッケージと **EnsPortal** パッケージも表示されますが、これらのパッケージは直接使用することを意図されていません。
ネームスペースが相互運用対応の場合、そのネームスペースにプロダクションを定義できます。詳細は、“相互運用プロダクションの概要”を参照してください。
- ・ このネームスペースで定義されたマッピングによって使用可能になる追加のコードまたはデータ。

コードは、拡張されたグローバル参照を使用して、他のネームスペースで定義されたグローバルにアクセスできます。詳細は、“グローバルの使用法”の“[グローバル構造](#)”を参照してください。

[InterSystems IRIS のセキュリティ・モデル](#)は、あらゆるユーザがどのデータにアクセスできるか、およびどのコードにアクセスできるかを制御します。

6.6.1 ネームスペースのシステム・グローバル

ネームスペースには、追加のシステム・グローバルが含まれ、それらは大まかに以下の 2 つのカテゴリに分類されます。

- ・ すべてのネームスペースに存在するシステム・グローバル。これらには、InterSystems IRIS によってユーザのルーチン、クラス定義、インクルード・ファイル、INT コード、および OBJ コードが格納されるグローバルが含まれます。
- ・ 特定の InterSystems IRIS 機能を使用するときに作成されるシステム・グローバル。例えば、ネームスペースで Analytics を使用する場合、独自の内部使用のためにシステムによって一連のグローバルが作成されます。

ほとんどの場合、これらのグローバルは、いずれも手動で書き込みしたり削除しないでください。“グローバルの使用法”の“[グローバルの名前付け規約](#)”を参照してください。

6.7 システム・ディレクトリ

どのネームスペースでも、ファイル・ストリームを作成すると、InterSystems IRIS によって既定のディレクトリにファイルが書き込まれ、それが後で削除されます。

このディレクトリへの書き込みアクセスがユーザに必要となるため、これは重要です。書き込みアクセスできない場合、コードでファイル・ストリームを作成できなくなります。

既定のディレクトリは、そのネームスペースのグローバル・データベースの **stream** サブディレクトリです。

6.8 詳細

このページで説明したトピックの詳細は、以下を参照してください。

- ・ [“グローバルの使用法”](#) には、拡張参照の使用を含む、グローバルの操作に関する情報が記載されています。
- ・ “専用のシステム/ツールおよびユーティリティ” の [“cvendian を使用したバイト・オーダー変換”](#) には、cvendian に関する情報が記載されています。
- ・ [“高可用性ガイド”](#) には、高可用性と復元可能性に関する情報が記載されています。
- ・ [“スケーラビリティ・ガイド”](#) には、分散キャッシングとシャーディングに関する情報が記載されています。

7

InterSystems IRIS セキュリティ

このページでは、インターシステムズのセキュリティの概要について説明し、特に InterSystems IRIS® アプリケーションの記述やメンテナンスに携わるプログラマに最も関係するトピックを詳しく取り上げます。

セキュリティの詳細は、“[インターシステムズのセキュリティについて](#)”を参照してください。

7.1 概要

このセクションでは、InterSystems IRIS 内でのセキュリティについて紹介し、InterSystems IRIS と外部システム間の通信について説明します。

7.1.1 InterSystems IRIS 内のセキュリティ要素

インターシステムズのセキュリティでは、以下の要素を基にしてシンプルで統合化されたアーキテクチャが提供されています。

- ・ **認証。** 認証とは、自分が自身で主張するとおりの人物であることを InterSystems IRIS に対して証明する手段です。信頼できる認証がないと、あるユーザが他人になりすまして不正に入手した特権を利用できるため、承認機能の重要性が損なわれることになります。

利用できる認証メカニズムは、InterSystems IRIS にアクセスする方法によって異なります。InterSystems IRIS には、使用可能な認証メカニズムがいくつかあります。プログラミングが必要なものもあります。
- ・ **承認。** ユーザが認証された後、セキュリティに関連する次の手順は、そのユーザに使用、閲覧、または変更が認められているリソースが何であるかを確認することです。この判断とそれに基づくアクセスの制御を、承認といいます。

プログラマの場合は、特定のユーザに特定のタスクを実行する許可があるかを確認するように、コードに適切なセキュリティ・チェックを実装する必要があります。
- ・ **監査。** 監査は、認証および承認システムのアクションなど、システムに関係する検証可能で信頼性の高い追跡のアクションを提供します。この情報は、セキュリティ関連の問題の後に発生する一連のイベントを再構成する基準となります。システムが監査されている事実を示すことが、攻撃者に対する抑止力として機能します（これは、攻撃を加えている間に自身の身元を知られてしまうことを、攻撃者が知っているからです）。

InterSystems IRIS では、監査可能な一連のイベントが提供されており、他のコードに追加できます。プログラマは、カスタム・イベントのコードに監査ログを含める必要があります。
- ・ **データベースの暗号化。** InterSystems IRIS データベースを暗号化することで、データを安全な状態で保護できます。暗号化により、承認されないユーザがデータベースの情報を見ることができないようにして、格納されている情報のセキュリティを確保します。InterSystems IRIS で実装している暗号化では AES (Advanced Encryption Standard)

アルゴリズムを採用しています。ディスクへの書き込みまたはディスクからの読み取りで、暗号化と解読が実行されます。暗号化と解読は最適化されていて、どのような InterSystems IRIS プラットフォームでも、暗号化と解読による影響は少なく、また判断可能です。実際、暗号化データベースへの書き込み時間が長くなることはまったくありません。

一般にデータベース暗号化のタスクで、コードの記述が必要になることはありません。

7.1.2 InterSystems IRIS との安全な接続

InterSystems IRIS と外部システム間で通信するとき、以下の追加のツールを使用できます。

- ・ SSL/TLS 構成。InterSystems IRIS では、SSL/TLS 構成を格納して、関連する名前を指定する機能がサポートされています。SSL/TLS 接続が必要な場合 (例えば HTTP 通信用)、プログラムによって該当する構成名を指定すると、InterSystems IRIS によって自動的に SSL/TLS 接続が処理されます。
- ・ X.509 証明書ストレージ。InterSystems IRIS では、X.509 証明書および秘密鍵をロードして、関連する構成名を指定する機能がサポートされています。X.509 証明書が必要な場合 (例えば SOAP をデジタル署名するため)、プログラムによって該当する構成名を指定すると、InterSystems IRIS によって自動的に証明書情報が抽出され使用されます。

オプションで、関連する秘密鍵ファイルにパスワードを入力したり、これを実行時に指定することができます。

- ・ 認証機関 (CA) へのアクセス。適切な形式の CA 証明書を所定の場所に配置している場合、InterSystems IRIS がこれを使用してデジタル署名などを検証します。

InterSystems IRIS は、CA 証明書を自動的に使用します。プログラミング作業は必要ありません。

7.2 InterSystems IRIS アプリケーション

ほぼすべてのユーザが、アプリケーションを使用して InterSystems IRIS と対話します。例えば、管理ポータル自体も一連のアプリケーションです。アプリケーションごとに、専用のセキュリティ機能があります。最も一般的な種類のアプリケーションは Web アプリケーションです。これは、Web ゲートウェイを介して InterSystems IRIS にアクセスするアプリケーションです。Web アプリケーションは、REST または SOAP 経由で Web ゲートウェイを介して通信します。

管理ポータルで、定義、変更および登録を行うことができます (十分な権限を持つユーザとしてログインした場合)。ただし、アプリケーションを導入するときには、インストールの一環でプログラムによってアプリケーションを定義の方が一般的です。InterSystems IRIS ではこれを実現する方法が提供されています。

InterSystems IRIS アプリケーションの詳細は、“[アプリケーション](#)” を参照してください。

7.3 InterSystems 認証モデル

プログラマの場合は、特定のユーザに特定のタスクを実行する許可があるかを確認するように、コードに適切なセキュリティ・チェックを実装する必要があります。このため、ロールベースのアクセスを使用するインターシステムズの認証モデルに精通する必要があります。この用語の概要は以下のとおりです。

- ・ アセット。アセットは、保護対象の項目です。アセットには、さまざまな種類があります。以下の項目は、すべてアセットです。
 - 各 InterSystems IRIS データベース
 - SQL を使用した InterSystems IRIS への接続機能
 - バックアップを実行する機能

- 各 Analytics KPI クラス
 - InterSystems IRIS で定義された各アプリケーション
- ・ リソース。リソースは、1 つまたは複数のアセットを関連付けることができる InterSystems IRIS のセキュリティ要素です。
- 一部のアセットの場合、アセットとリソース間の関連付けは、構成オプションです。データベースを作成するときは、関連付けられたリソースを指定します。同様に、InterSystems IRIS アプリケーションを作成するときは、関連付けられたリソースを指定します。
- その他のアセットは、関連付けがハードコーディングされています。Analytics KPI クラスの場合、関連付けられたリソースをそのクラスのパラメータとして指定します。
- ユーザ定義のアセットおよびリソースの場合、ハードコーディングするか適当な構成システムを定義するか of the いくつかの方法で、自由に関連付けを行うことができます。
- ・ ロール。ロールは、名前と一連の関連付けられた権限 (大量になる場合があります) を指定するインターシステムズのセキュリティ要素です。権限は、特定のリソースについての特定のタイプ (読み込み、書き込み、使用など) の許可です。例えば、以下は権限です。
- データベースを読み込む許可
 - テーブルに書き込む許可
 - アプリケーションを使用する許可
- ・ ユーザ名。ユーザ名 (または略してユーザ) は、ユーザが InterSystems IRIS にログオンするときに使用するインターシステムズのセキュリティ要素です。各ユーザは、1 つまたは複数のロールに属します (またはメンバになります)。

その他の重要な概念は、ロール・エスカレーションです。一時的に 1 つ以上の新規ロールをユーザに (プログラムによって) 追加して、特定のコンテキストでは通常は許可されないタスクを実行可能にすることが必要になります。これを、ロール・エスカレーションといいます。ユーザがそのコンテキストを終了した後で、一時的ロールを削除します。これをロールのエスカレーション解除と呼びます。

管理ポータルで、リソース、ロール、およびユーザの定義、変更および削除を行うことができます (十分な権限を持つユーザとしてログインした場合)。ただし、アプリケーションを導入するときには、インストールの一環でプログラムによってリソース、ロール、および開始時のユーザ名を定義する方が一般的です。InterSystems IRIS ではこれを実現する方法が提供されています。

8

ローカライズのサポート

このページでは、InterSystems IRIS® のローカライズのサポートの概要を説明します。

8.1 概要

InterSystems IRIS はローカライズをサポートしています。これにより、アプリケーションを再設計せずに複数の国または複数の地域向けのアプリケーションを開発できます。ローカライズ・モデルは、以下のように動作します。

- ・ InterSystems IRIS には、一連の定義済みロケールが用意されています。InterSystems IRIS のロケールとは、ユーザーの言語、通貨記号、書式、および特定の国または地域に固有のその他の規約を指定するメタデータのセットです。
ロケールは、InterSystems IRIS データベースに書き込むときに使用する文字エンコードを指定します。また、他の文字エンコードとの間で文字変換を処理するために必要な情報も含まれます。
- ・ InterSystems IRIS サーバをインストールするときに、サーバの既定のロケールがインストーラによって設定されます。
インストール後に既定のロケールを変更することはできませんが、必要に応じて、既定以外のロケールを使用するように指定できます。
- ・ 管理ポータルには、ブラウザ設定で固定の言語セットに指定されたローカル言語の文字列が表示されます。
- ・ 独自のアプリケーションに、ローカライズされた文字列を指定できます。[“文字列のローカライズとメッセージ・ディクショナリ”](#)を参照してください。このメカニズムは、REST サービスおよび Business Intelligence 要素に使用できます。

8.2 InterSystems IRIS のロケールと各国言語のサポート

InterSystems IRIS のロケールは、特定の国または地域に適用されるストレージと表示の規約を定義するメタデータのセットです。ロケールの定義には、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
- ・ 標準 (ISO、Unicode、またはその他) によって定義される既定の文字セット (このロケールの文字エンコード)。

InterSystems IRIS では、文字セットと文字エンコードの語句を同義語として使用しますが、厳密には必ずしも同義ではありません。

- 他のサポートされている文字セットとの間で文字を変換する変換テーブル (I/O テーブルとも呼ぶ) のセット。

特定の文字セット (CP1250 など) の “変換テーブル” は、実際にはテーブルのペアです。1 つのテーブルは既定の文字セットを他言語文字セットに変換する方法を指定し、もう 1 つのテーブルは反対方向に変換する方法を指定します。InterSystems IRIS では、このテーブルのペアを 1 つの単位として参照して変換を実行します。

InterSystems IRIS は、各国言語サポート (NLS) という語句を使用して、ロケール定義を表示して拡張するために使用するツールとロケール定義を一括して示します。

管理ポータルには、既定のロケールを確認し、インストールされているロケールの詳細を表示してロケールを表示するページがあります。以下に例を示します。

Locale properties of enuw (English, United States, Unicode):
 Your current locale is: enuw (English, United States, Unicode)
 (enuw is a system locale. Edit is not allowed.)

Basic Properties

Name	Value
Country	United States (US)
Language	English (en-US)
Character set	Unicode
Currency	\$

このページを使用して、利用可能な変換テーブルの名前を確認することもできます。それらの名前は InterSystems IRIS に固有のものです。(これらのテーブルの名前がわかっていることが必要な場合があります。)

この管理ポータル・ページにアクセスして使用方法の詳細は、“システム管理ガイド”の“[管理ポータルの NLS ページの使用法](#)”を参照してください。

InterSystems IRIS には、一連のクラスも用意されています (%SYS.NLS パッケージおよび Config.NLS パッケージ内)。“専用のシステム/ツールおよびユーティリティ”の“[各国言語サポートのシステム・クラス](#)”を参照してください。

8.3 既定の入出力テーブル

ロケール定義の外側で、特定の InterSystems IRIS インスタンスは、入出力アクティビティに特定の変換テーブルを使用するように既定で構成されています。具体的には、次の各シナリオで既定の変換テーブルを使用するように指定されています。

- InterSystems IRIS プロセスとの通信時
- InterSystems ターミナルとの通信時
- ファイルの読み取りおよび書き込み時
- TCP/IP デバイスの読み取りおよび書き込み時
- オペレーティング・システムにパラメータ (ファイル名やパスなど) として送信される文字列の読み取りおよび書き込み時

- ・ プリンタなどのデバイスの読み取りおよび書き込み時

例えば、パラメータ (ファイル名やパスなど) として文字列を受け取るオペレーティング・システム関数を InterSystems IRIS から呼び出す必要がある場合、まず、呼び出し先の `syscall` に適した NLS 変換に文字列を渡します。この変換結果をオペレーティング・システムに送信します。

現在の既定値を確認するには、`%SYS.NLS.Table` を使用します。詳細は、[クラス・リファレンス](#)を参照してください。

8.4 ファイルと文字エンコード

データベースの外部にあるエンティティに対する読み取りまたは書き込みを行うときには、そのエンティティが InterSystems IRIS とは別の文字セットを使用している可能性が常に存在します。最も一般的なシナリオは、ファイルの操作です。

最も低いレベルでは、[Open](#) コマンドを使用してファイルまたは他のデバイスを開きます。このコマンドは、文字をデバイス間で変換するときに使用する変換テーブルを指定したパラメータを受け入れることができます。詳細は、["入出力デバイス・ガイド"](#)を参照してください。InterSystems IRIS は、必要に応じてそのテーブルを使用して文字を変換します。

同様に、オブジェクト・ベースのファイル API を使用する場合は、ファイルの `TranslateTable` プロパティを指定します。

(プロダクション・アダプタ・クラスには、他言語文字セットを指定するためのこれに代わるプロパティがあり、入力データとして想定している文字エンコード・システムおよび出力データで目的とする文字エンコードとして使用されます。この場合、インターシステムズでサポートしているセットから選択して標準の文字セット名を指定します。)

8.5 文字の手動変換

InterSystems IRIS の [\\$ZCONVERT](#) 関数を使用すると、文字を別の文字セットとの間で手動で変換できます。

9

サーバ構成オプション

コードの作成方法に影響を与えるサーバの構成オプションがいくつかあります。

構成詳細の大部分は、`iris.cpf` というファイル（構成パラメータ・ファイル、つまり CPF）に保存されます。

9.1 InterSystems SQL の設定

このセクションでは、InterSystems SQL の動作に影響する最も重要な設定について説明します。

9.1.1 クエリキャッシュのソースを保持

この設定は、埋め込み SQL 以外の InterSystems SQL を実行するときに、InterSystems IRIS によって生成されるルーチンおよび INT コードを保存するかどうかを指定します。どの場合でも、生成された OBJ は保持されます。既定では、ルーチンおよび INT コードは保持されません。

クエリ結果は、キャッシュに格納されません。

この設定を変更するには以下の手順に従います。

1. 管理ポータルにログインします。
2. [システム管理]→[構成]→[SQL とオブジェクトの設定]→[SQL] の順に選択します。
3. [SQL] ページで、必要に応じて [クエリキャッシュのソースを保持] チェック・ボックスにチェックを付けるか、チェックを外します。
4. [保存] をクリックします。

9.1.2 デフォルトスキーマ

この設定は、スキーマが指定されていないテーブルを作成または削除するときに使用する既定のスキーマの名前を指定します。ビュー、トリガ、またはストアド・プロシージャの作成や削除など、その他の DDL オペレーションでもこの設定を使用します。

この設定を変更するには以下の手順に従います。

1. 管理ポータルにログインします。
2. [システム管理]→[構成]→[SQL とオブジェクトの設定]→[SQL] の順に選択します。
3. [SQL] ページで、必要に応じて [デフォルトスキーマ] フィールドを編集します。

4. **【保存】** をクリックします。

詳細は、“[スキーマ名](#)” を参照してください。

9.1.3 区切り識別子のサポート

この設定は、二重引用符で囲まれた文字を InterSystems SQL でどのように扱うかを制御します。

区切り識別子のサポートを有効 (既定の設定) にすると、フィールドの名前を二重引用符で囲むことができます。これにより、標準的な識別子ではない名前を持つフィールドを参照できるようになります。そのようなフィールドの例としては、名前に SQL 予約語を使用しているものがあります。

区切り文字のサポートを無効にすると、二重引用符内の文字は文字列リテラルとして処理され、標準的な識別子ではない名前を持つフィールドを参照することはできなくなります。

SET OPTION コマンドに `SUPPORT_DELIMITED_IDENTIFIERS` キーワードを指定するか、`$SYSTEM.SQL.Util.SetOption()` メソッドの `DelimitedIdentifiers` オプションを使用することで、システム全体に区切り識別子のサポートを設定できます。現在の設定を確認するには、`$SYSTEM.SQL.CurrentSettings()` を呼び出します。

詳細は、“[区切り識別子](#)” を参照してください。

9.2 IPv6 アドレスの使用

InterSystems IRIS® では、常に IPv4 アドレスおよび DNS 形式のアドレス指定に対応します (ホスト名、ドメイン修飾子ありおよびなし)。また、InterSystems IRIS を IPv6 アドレスに対応するように構成できます。“[システム管理ガイド](#)” の “[IPv6 のサポート](#)” を参照してください。

9.3 プログラムによるサーバの構成

特定のユーティリティを呼び出すことで InterSystems IRIS の運用パラメータのいくつかをプログラムで変更できます。顧客の構成を変更する際はこの方法を使用することがあります。以下はその例です。

- ・ **Config.Miscellaneous** には、システム全体の既定の動作と設定を指定するメソッドがあります。
- ・ **%SYSTEM.Process** には、現在のプロセスの有効期間中の環境値を設定するメソッドがあります。
- ・ **%SYSTEM.SQL** には、SQL 設定を変更するためのメソッドがあります。

詳細は、これらのクラスのインターシステムズ・クラス・リファレンスを参照してください。

9.4 詳細

このページで説明したトピックの詳細は、以下を参照してください。

- ・ “[システム管理ガイド](#)” には、管理ポータルの大部分の使用方法が記載されています。
- ・ “[インターシステムズ・クラス・リファレンス](#)” には、InterSystems IRIS で提供されるすべての非内部クラスに関する詳細があります。
- ・ “[構成パラメータ・ファイル・リファレンス](#)” には、CPF のリファレンス情報が記載されています。

10

効果的なスキル

このページでは、プログラマが実行方法を理解していると効果的な特定のタスクについて簡単に説明します。ここで説明するタスクを実行する方法、条件、理由に精通していれば作業時間と作業量を節減できます。

10.1 データベースの定義

ローカル・データベースを作成するには以下の手順に従います。

1. 管理ポータルにログインします。
2. [システム管理]→[構成]→[システム構成]→[ローカルデータベース] を選択します。
3. [新規データベース作成] を選択し、データベース・ウィザードを開始します。
4. 新規データベースに関する以下の情報を入力します。
 - ・ テキスト・ボックスにデータベース名を入力します。普通は英数字を使用した短い文字列を指定します。ルールの詳細は、“[データベースの構成](#)”を参照してください。
 - ・ ディレクトリ名を入力するか、[参照] を選択してデータベース・ディレクトリを選択します。作成する最初のデータベースである場合は、データベースの作成先とする親ディレクトリを参照して選択する必要があります。他のデータベースを既に作成している場合は、最後に作成したデータベースの親ディレクトリが既定のデータベース・ディレクトリになります。
5. [完了] を選択します。

リモート・データベースの作成に関する追加のオプションと詳細は、“[IRIS の構成](#)”を参照してください。

10.2 ネームスペースの定義

ローカル・データベースを使用するネームスペースを作成するには以下の手順に従います。

1. 管理ポータルにログインします。
2. [システム管理]→[構成]→[システム構成]→[ネームスペース] を選択します。
3. [新規ネームスペース作成] を選択します。

4. [名前] にネームスペース名を入力します。普通は英数字を使用した短い文字列を指定します。ルールの詳細は、“[ネームスペースの構成](#)”を参照してください。
5. [グローバルのための既存のデータベースを選択] でデータベースを選択するか、[新規データベース作成] を選択します。
[新規データベース作成] を選択すると、[データベースを作成](#)する場合と同様のオプションが表示されます。
6. [ルーチンのための既存のデータベースを選択] でデータベースを選択するか、[新規データベース作成] をクリックします。
[新規データベース作成] を選択すると、[データベースを作成](#)する場合と同様のオプションが表示されます。
7. [保存] を選択します。

追加のオプションの詳細は、“[IRIS の構成](#)”を参照してください。

10.3 グローバルのマッピング

データベース ABC にグローバルをマッピングする場合、各自のネームスペースで既定のデータベースではないデータベース ABC に対して IRIS がこのグローバルを読み書きできるように、所定のネームスペースを構成します。このグローバル・マッピングを定義すると、グローバルが既に存在している場合、指定のデータベースにはそのグローバルが移動しなくなります。以降は、グローバルを読み書きする場所がマッピングから IRIS に指示されます。

グローバルをマッピングするには以下の手順に従います。

1. グローバルが既に存在する場合は、それを目的のデータベースに移動します。“[1つのデータベースから別のデータベースへのデータの移動](#)”を参照してください。
2. 管理ポータルにログインします。
3. [システム管理]→[構成]→[システム構成]→[ネームスペース] を選択します。
4. このマッピングの定義先とする、ネームスペースの行で [グローバルマッピング] を選択します。
5. [新規グローバルマッピング] を選択します。
6. [グローバルデータベース位置] で、このグローバルを格納するデータベースを選択します。
7. [グローバル名] に入力します (先頭のキャレットは省略します)。* 文字を使用して複数のグローバルを選択できます。

このグローバルは、マッピングの時点で存在していなくてもかまいません (つまり、これから作成するグローバルの名前を使用できます)。

注釈 一般的には永続クラスのデータ・グローバルのマッピングを作成します。そのデータを既定以外のデータベースに格納するからです。多くの場合、データ・グローバルの名前は推測できますが、名前が長すぎると、ハッシュ化した形式のクラス名が自動的に使用される点に留意してください。これらのクラスのストレージ定義を確認して、そのクラスで使用されるグローバルの正確な名前を把握しておくことは無駄ではありません。“[ストレージ](#)”を参照してください。

8. [OK] をクリックします。
9. マッピングを保存するには [変更を保存] をクリックします。

詳細は、“[システム管理ガイド](#)”を参照してください。

また、グローバル・マッピングをプログラムで定義することもできます。“[インターシステムズ・プログラミング・ツールの索引](#)”のグローバルのエントリを参照してください。

以下の図は、管理ポータルに表示したグローバル・マッピングの例です。ここでは、グローバル名先頭のキャレットが表示されていません。

The global mappings for namespace NOTES are displayed below:

Filter:	Page size: 0	Max rows: 1000	Results: 1	Page: < « 1 » > of 1
Global	Subscript	Database		
MyMappedGlobal		CACHETEMP	Edit	Delete

このマッピングは以下の処理を表しています。

- ・ ネームスペース DEMONAMESPACE で、グローバル ^MyTempGlobal のノードの値を設定する場合は CACHETEMP データベースにデータが書き込まれます。

そのノードを直接設定するか、オブジェクト・クラスまたは SQL を介して間接的に設定するかに関係なく、この動作になります。

- ・ ネームスペース DEMONAMESPACE で、グローバル ^MyTempGlobal から値を取得する場合は CACHETEMP データベースからデータが読み取られます。

そのノードの値を直接取得するか、オブジェクト・クラスまたは SQL を介して間接的に取得するかに関係なく、この動作になります。

10.4 ルーチンのマッピング

データベース ABC にルーチンをマッピングする場合、各自のネームスペースで既定のデータベースではないデータベース ABC で IRIS がこのルーチンを探し出すことができるように、所定のネームスペースを構成します。このルーチン・マッピングを定義すると、ルーチンが既に存在している場合、指定のデータベースにはそのルーチンが移動しなくなります。以降は、ルーチンを探す場所がマッピングから IRIS に指示されます。

ルーチンをマッピングするには以下の手順に従います。

1. ルーチンが既に存在する場合は、それをエクスポートしてインポートすることにより、目的のデータベースにそのルーチンをコピーします。
2. 管理ポータルにログインします。
3. [システム管理]→[構成]→[システム構成]→[ネームスペース]を選択します。
4. このマッピングの定義先とする、ネームスペースの行で [ルーチンマッピング] を選択します。
5. [新規ルーチンマッピング] を選択します。
6. [ルーチンデータベース位置] で、このルーチンを格納するデータベースを選択します。
7. [ルーチン名] に値を入力します。* 文字を使用して複数のルーチンを選択できます。

実際のルーチン名を使用します。つまり、ルーチン名の先頭にはキャレット (^) を記述しません。

このルーチンは、マッピングの時点で存在していなくてもかまいません (つまり、これから作成するルーチンの名前を使用できます)。

8. [ルーチンタイプ] を選択します。
9. [OK] をクリックします。

10. [OK] をクリックします。

11. マッピングを保存するには **[変更を保存]** をクリックします。

詳細は、“[システム管理ガイド](#)” を参照してください。

この種類のマッピングをプログラムで定義することもできます。また、ルーチン・マッピングをプログラムで定義することもできます。“[インターシステムズ・プログラミング・ツールの索引](#)” のルーチンのエントリを参照してください。

重要 1 つ以上のルーチンをマップするときには、それらのルーチンが必要とするすべてのコードとデータを必ず特定して、そのすべてのコードとデータがすべてのターゲット・ネームスペースで使用できることを確認します。マップされるルーチンは、以下の項目に依存している可能性があります。

- ・ インクルード・ファイル
- ・ その他のルーチン
- ・ クラス
- ・ テーブル
- ・ グローバル

追加のルーチン、パッケージ、およびグローバル・マッピングを必要に応じて使用して、これらの項目がターゲット・ネームスペースで使用できるようにします。

10.5 パッケージのマッピング

データベース ABC にパッケージをマッピングする場合、各自のネームスペースで既定のデータベースではないデータベース ABC で、IRIS がこのパッケージのクラス定義を探し出すことができるように、所定のネームスペースを構成します。このクラス定義に関連付けて生成したルーチンにも、このマッピングが適用されます。これらのルーチンは同じパッケージに存在します。このマッピングは、これらのパッケージで永続クラスに関連して格納されているデータの場所には影響しません。

また、このパッケージ・マッピングを定義すると、パッケージが既に存在している場合、指定のデータベースにはそのパッケージが移動しなくなります。以降は、パッケージを探す場所がマッピングから IRIS に指示されます。

パッケージをマッピングするには以下の手順に従います。

1. パッケージが既に存在する場合は、それをエクスポートしてインポートすることにより、目的のデータベースにそのパッケージをコピーします。
2. 管理ポータルにログインします。
3. **[システム管理]**→**[構成]**→**[システム構成]**→**[ネームスペース]** を選択します。
4. このマッピングの定義先とする、ネームスペースの行で **[パッケージマッピング]** を選択します。
5. **[新規パッケージマッピング]** を選択します。
6. **[パッケージデータベース位置]** で、このパッケージを格納するデータベースを選択します。
7. **[パッケージ名]** に値を入力します。

このパッケージは、マッピングの時点で存在していなくてもかまいません（つまり、これから作成するパッケージの名前を使用できます）。

8. [OK] をクリックします。
9. [OK] をクリックします。

10. マッピングを保存するには **[変更を保存]** をクリックします。

詳細は、“[システム管理ガイド](#)” を参照してください。

この種類のマッピングをプログラムで定義することもできます。また、パッケージ・マッピングをプログラムで定義することもできます。“[インターシステムズ・プログラミング・ツールの索引](#)” のパッケージのエントリを参照してください。

重要 パッケージをマップするときには、そのパッケージ内のクラスが必要とするすべてのコードとデータを特定して、そのすべてのコードとデータがすべてのターゲット・ネームスペースで使用できることを必ず確認します。マップされるクラスは、以下の項目に依存している可能性があります。

- ・ インクルード・ファイル
- ・ ルーチン
- ・ その他のクラス
- ・ テーブル
- ・ グローバル

追加のルーチン、パッケージ、およびグローバル・マッピングを必要に応じて使用して、これらの項目がターゲット・ネームスペースで使用できるようにします。

10.6 テスト・データの生成

IRIS は、永続クラスで使用する擬似ランダム・テスト・データを作成するユーティリティを備えています。このようなデータの作成をデータ生成と呼び、データ生成を実行するユーティリティを `Populate` ユーティリティと呼びます。大量のデータを扱っている場合、アプリケーションのさまざまな部分がどのように機能するかをテストする場合に、このユーティリティが特に効果的です。

`Populate` ユーティリティは、`%Library.Populate` と `%Library.PopulateUtils` の 2 つのクラスで構成されます。これらのクラスは、さまざまな一般的な形式のデータを生成するメソッドを提供します。例えば、ランダム名を生成する以下のメソッドがあります。

ObjectScript

```
Write ##class(%Library.PopulateUtils).Name()
```

以下の 2 つの方法で `Populate` ユーティリティを使用できます。

10.6.1 %Populate の拡張

この方法では以下の手順を実行します。

1. 使用するクラスのスーパークラス・リストに `%Populate` を追加します。
2. 必要に応じて、クラスの各プロパティの `POPSPEC` パラメータに値を指定します。

このパラメータの値には、プロパティ値としての使用に適した値を返すメソッドを指定します。

以下に例を挙げます。

Class Member

```
Property SSN As %String(POPSPEC = "##class(MyApp.Utils).MakeSSN()");
```

3. 適切な順序 (先に独立クラス、次に依存クラス) でデータを生成するユーティリティ・メソッドまたはルーチンを作成します。

このコードで、クラスを生成するには、そのクラスの `Populate()` メソッドを実行します。このメソッドは `%Populate` スーパークラスから継承します。

このメソッドによって、目的のクラスのインスタンスが生成され、`%Save()` メソッドを呼び出すことで保存されます。これにより、各プロパティを検証してから保存できます。

プロパティごとに、このメソッドによって次のように値が生成されます。

- a. プロパティに `POPSPEC` パラメータを指定している場合は、このメソッドが呼び出され、返された値が使用されます。
- b. そのように指定していない場合、プロパティ名が `City`、`State`、`Name` などの事前定義された値であれば、その値に適したメソッドが呼び出されます。これらの値はハードコーディングされています。
- c. 上記以外の場合はランダムな文字列が生成されます。

`%Populate` クラスでシリアル・プロパティやコレクションなどがどのように扱われるかの詳細は、“[Populate ユーティリティの使用](#)”を参照してください。

4. ターミナルからユーティリティ・メソッドを呼び出します。適切な任意の起動コードから呼び出す場合もあります。

これは、`SAMPLES` データベースの `Sample.Person` に使用する一般的な方法です。

10.6.2 %Populate および %PopulateUtils のメソッドの使用

`%Populate` クラスと `%PopulateUtils` クラスは、特定の形式の値を生成するメソッドを提供します。以下に示す代替のデータ生成手法では、これらのメソッドを直接呼び出すことができます。

1. 適切な順序 (先に独立クラス、次に依存クラス) でデータを生成するユーティリティ・メソッドを作成します。

このコードでは、各クラスに対して目的の回数だけ繰り返し処理が実行されます。繰り返し処理ごとに、以下の手順を実行します。

- a. 新しいオブジェクトを作成します。
- b. 適切なランダム値 (または、ランダムに近い値) を使用して各プロパティを設定します。
そのためには、`%Populate` または `%PopulateUtils` のメソッドを使用するか、独自に作成したメソッドを使用します。
- c. オブジェクトを保存します。

2. 作成したユーティリティ・メソッドをターミナルから呼び出します。

これは `SAMPLES` データベースの 2 つの `DeepSee` サンプルに使用方法です。このサンプルは、`DeepSee` パッケージと `HoleFoods` パッケージにあります。

10.7 格納したデータの削除

開発プロセスでは、クラスの既存のテスト・データをすべて削除してから再生成することが必要になる場合があります (ストレージ定義を削除した場合など)。

クラスの保存済みデータを簡単に削除できる 2 つの方法を以下に示します (さらに別の方法もあります)。

- ・ 以下のクラス・メソッドを呼び出します。

```
##class(%ExtentMgr.Util).DeleteExtent(classname)
```

classname は、完全なパッケージとクラス名です。

- ・ クラスのデータとクラスのインデックスが格納されているグローバルを削除します。以下のように管理ポータルから実行する方が簡単な場合もあります。

1. [システムエクスプローラ]→[グローバル] を選択します。
2. [削除] を選択します。
3. 左側で、作業中のネームスペースを選択します。
4. 右側で、データ・グローバルとインデックス・グローバルの横にあるチェック・ボックスにチェックを付けます。
5. [削除] を選択します。

これらのグローバルを削除してもよいかどうか確認を求められます。

これらのオプションによってデータは削除されますが、クラス定義は削除されません (逆にクラス定義を削除しても、データは削除されません)。

10.8 ストレージのリセット

重要

開発段階でストレージをリセットできることは重要ですが、ライブ・システムでストレージをリセットすることはありません。

クラスのストレージをリセットすると、クラスがその保存済みデータにアクセスする方法が変わります。クラスに保存済みデータがある場合に、プロパティ定義を削除、追加、または変更してからストレージをリセットすると、その保存済みデータに正しくアクセスできなくなることがあります。したがって、ストレージをリセットする場合は、そのクラスの既存のデータもすべて削除し、必要に応じてそれを再生成または再ロードする必要があります。

IDE でクラスのストレージをリセットするには以下の手順に従います。

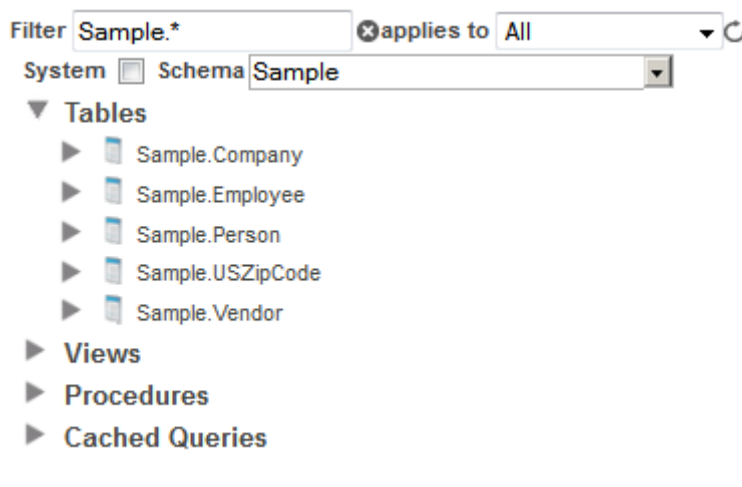
1. クラスを表示します。
2. クラス定義の末尾までスクロールします。
3. ストレージ定義全体 (<Storage name= から </Storage> まで) を選択します。選択した内容を削除します。
4. クラスを保存し、リコンパイルします。

10.9 テーブルの参照

テーブルを参照するには、管理ポータルで以下の手順を実行します。

1. [システムエクスプローラ]→[SQL] を選択します。
2. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
3. [スキーマ] ドロップダウン・リストから必要に応じて SQL スキーマを選択します。このリストには、このネームスペースにある SQL スキーマがすべて表示されます。各スキーマは、最上位のクラス・パッケージに相当します。

4. このスキーマのすべてのテーブルを表示するには [テーブル] フォルダを展開します。例えば、以下のようになります。



5. テーブルの名前を選択します。右側の領域に、そのテーブルに関する情報が表示されます。
6. [テーブルを開く] を選択します。

このテーブルの先頭から 100 行が表示されます。例えば、以下のようになります。

Refresh Close Window

Sample.Person in namespace SAMPLES

#	ID	Age	DOB	FavoriteColors	Name	SSN	Spouse	Home_City	Home_State	Home_Street
1	1	14	03/20/2000	Red	Newton,Dave R.	384-10-6538		Pueblo	AK	6977 First Street
2	2	17	05/30/1997	Green	Waterman,Danielle C.	944-39-5991		Oak Creek	ID	1648 Maple Street
3	3	86	04/01/1928		DeSantis,Christen N.	336-13-6311		Boston	AZ	8572 Maple Street
4	4	55	02/29/1960	Purple	Baker,Marvin Z.	198-22-7709		Queensbury	NV	1243 First Blvd
5	5	80	12/13/1934	Black	Diavolo,Ralph A.	586-13-9662		Hialeah	NY	3880 Maple Pl
6	6	38	10/13/1976		Russell,Paul S.	572-40-8824		Denver	CA	7269 Main Pl
7	7	33	11/26/1981	Purple Purple	Pascal,John X.	468-82-7179		Zanesville	AR	872 Elm Street

以下の点に注意してください。

- ここに表示される値は、表示値であり、ディスクに格納されている論理値ではありません。
- 最初の列 (#) は、表示されている順番による行番号です。
- 2 番目の列 ([ID]) は、このテーブルの行に対する一意の識別子であり、このクラスのオブジェクトを開くときは、この識別子を使用します(このクラスではこれらの識別子は整数ですが、整数ではないこともあります)。

このテーブルは、SAMPLES データベースが構築されるたびに新しく生成されるため、この場合、これらの番号は同一になっています。実際のアプリケーションでは、いくつかのレコードが削除されていることがあります。その場合は [ID] の値が不連続になり、左の行番号と一致しくなくなります。

10.10 SQL クエリの実行

SQL クエリを実行するには、管理ポータルで以下の手順を実行します。

- [システムエクスプローラ]→[SQL] を選択します。

2. 必要に応じ、ヘッダ領域で **[変更]** を選択して目的のネームスペースを選択します。
3. **[クエリ実行]** を選択します。
4. 入力ボックスに SQL クエリを入力します。以下に例を挙げます。

```
select * from sample.person
```

5. ドロップダウン・リストで、**[表示モード]**、**[論理モード]**、または **[ODBC モード]** を選択します。
これにより、ユーザ・インタフェースに結果をどのように表示するかを制御します。
6. 次に **[実行]** を選択します。ポータルに結果が表示されます。以下に例を挙げます。

Row count: 200 Performance: 0.015 seconds 3442 global references Cached Query: [%sqlcq.SAMPLES.cls21](#) Last update: 2

ID	Age	DOB	FavoriteColors	Name	SSN	Spouse	Home_City	Home_State	Home_Stree
1	34	50813		Ott,Liza F.	126-19-4431		Islip	OH	7433 Second Court
2	69	37939		Ingrahm,Sally N.	896-94-8820		Islip	IA	6707 Franklin Court
3	40	48588	OrangeGreen	Eagleman,Angela N.	937-68-7407		Denver	AL	4440 Madison Court
4	23	54736	Yellow	Ingersol,Umberto S.	381-48-8952		St Louis	WV	9908 Oak Blvd
5	11	59217		Mara,George F.	956-42-9085		Elmhurst	NE	5290 Ash Drive

10.11 オブジェクト・プロパティの検証

ターミナルでオブジェクトを開いて特定のプロパティ名を入力することで、そのプロパティの値を最も容易に表示できることがあります。

1. ターミナル・プロンプトが目的のネームスペースの名前ではない場合は、以下のように入力して Enter キーを押します。

```
ZN "namespace"
```

namespace は、目的のネームスペースの名前です。

2. 以下のようなコマンドを入力して、このクラスのインスタンスを開きます。

```
set object=##class(package.class).%OpenId(ID)
```

package.class はパッケージおよびクラスの名前で、ID はクラスに保存したオブジェクトの ID です。

3. 以下のように入力してプロパティの値を表示します。

```
write object.propname
```

propname は、表示する値を収めたプロパティの名前です。

10.12 グローバルの表示

一般的にグローバルを表示するには、ObjectScript の **ZWRITE** コマンドまたは管理ポータルの **[グローバル]** ページを使用します。クラスのデータを収めたグローバルを探す場合は、まずクラス定義を確認して、表示するグローバルを把握しておくことが効果的です。

1. 特定のクラスのデータ・グローバルを探していて、そのクラスのデータがどのグローバルに格納されるかわからない場合は、以下の手順を実行します。
 - a. IDE でクラスを表示します。
 - b. クラス定義の末尾までスクロールします。
 - c. `<DefaultData>` 要素を探します。`<DefaultData>` と `</DefaultData>` の間の値が、このクラスのデータを格納しているグローバルの名前です。

IRIS では、このようなグローバルの名前を決定する際に簡潔な命名規則が使用されます。詳細は、“[永続クラスによって使用されるグローバル](#)”を参照してください。ただし、グローバル名は最大 31 文字（先頭のキャレットを除く）であることから、完全なクラス名が長い場合、代わりにハッシュ化した形式のグローバル名が自動的に使用されます。

2. 管理ポータルで、**[システムエクスプローラ]**→**[グローバル]** を選択します。
3. 必要に応じ、ヘッダ領域で **[変更]** を選択して目的のネームスペースを選択します。

このネームスペースに存在するグローバルのリストがポータルに表示されます（それぞれの名前では先頭のキャレットが省略されていることがわかります）。以下に例を挙げます。

Page size: 0 Results: 84 Page: 1 of 1

<input type="checkbox"/> Name	Location	Keep	Collation	
<input type="checkbox"/> Aviation.AircraftD	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.AircraftI	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.Countries	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.CrewI	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.EventD	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.EventI	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Aviation.States	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> CacheMsg	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit
<input type="checkbox"/> Cinema.ReviewD	c:\intersystems\ensemble\mgr\samples\	No	Cache standard	View Edit

通常、システム・グローバルではないほとんどのグローバルは永続クラスのデータを格納しています。つまり、システム・グローバルを表示しない限り、ほとんどのグローバルは見慣れた名前のグローバルです。

4. 目的のグローバルの行で **[表示]** を選択します。

このグローバルの先頭から 100 個のノードが表示されます。以下に例を挙げます。

Global Search Mask: ^Sample.PersonD [Display] [Cancel]

Search History: ^Sample.PersonD [v] [x] Maximum Rows: 100 [Allow Edit]

1: ^Sample.PersonD = 200

2: ^Sample.PersonD(1) = \$1b("","Newton,Dave R.,"384-10-6538",58153,\$1b("6977 First Street","Pueblo","AR

3: ^Sample.PersonD(2) = \$1b("","Waterman,Danielle C.,"944-39-5991",57128,\$1b("1648 Maple Street","Oak C

4: ^Sample.PersonD(3) = \$1b("","DeSantis,Christen N.,"336-13-6311",31867,\$1b("8572 Maple Street","Bost

5: ^Sample.PersonD(4) = \$1b("","Baker,Marvin Z.,"198-22-7709",43523,\$1b("1243 First Blvd","Queensbury",

5. 目的のオブジェクトのみとなるように表示を絞り込むには、[グローバル検索マスク] フィールドで、グローバル名の末尾に (ID) (オブジェクトの ID) を付加します。以下に例を挙げます。

```
^Sample.PersonD(45)
```

つづいて [表示] を選択します。

前述のように、ZWRITE コマンドを使用することもできます。このコマンドの省略形である “ZW” を使用することもできます。ターミナルで以下のようにコマンドを入力します。

```
zw ^Sample.PersonD(45)
```

10.13 クエリのテストとクエリ・プランの表示

コードで実行するクエリを管理ポータルでテストできます。そこからクエリ・プランを表示することもできます。クエリ・プランでは、クエリ・オブティマイザでクエリがどのように実行されるかに関する情報が得られます。この情報を使用して、クラスにインデックスを追加する必要があるか、別の方法でクエリを記述する必要があるかなどを判断できます。

クエリ・プランを表示するには、管理ポータルで以下の手順を実行します。

1. [システムエクスプローラ]→[SQL] を選択します。
2. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
3. [クエリ実行] を選択します。
4. 入力ボックスに SQL クエリを入力します。以下に例を挙げます。

```
select * from sample.person
```

5. ドロップダウン・リストで、[表示モード]、[論理モード]、または [ODBC モード] を選択します。
これにより、ユーザ・インタフェースに結果をどのように表示するかを制御します。
6. クエリをテストするには [実行] を選択します。
7. クエリ・プランを表示するには [プラン表示] を選択します。

10.14 クエリ・キャッシュの表示

IRIS SQL では、埋め込み SQL として使用する場合を除き、データにアクセスするための再使用可能なコードが生成され、そのコードがクエリ・キャッシュに配置されます (埋め込み SQL では、再使用可能なコードは生成されますが、生成された INT コードの中に保持されます)。

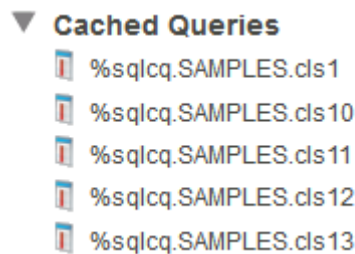
その SQL 文を初めて実行すると IRIS によってクエリが最適化され、データを取得するコードが生成されて格納されます。このコードは、最適化されたクエリ・テキストと共にクエリ・キャッシュに格納されます。このキャッシュは、OBJ コードのキャッシュであり、データのキャッシュではありません。

この SQL 文を再度実行すると、IRIS によって最適化され、そのクエリのテキストとクエリ・キャッシュの項目が比較されます。格納されているクエリが、指定されたクエリと一致していると IRIS によって判断されると（空白などのわずかな相違を除く）、そのクエリに関して格納されているコードが使用されます。

管理ポータルでは、クエリ・キャッシュの項目がスキーマ別にグループ化されます。特定のスキーマのクエリ・キャッシュを表示するには、管理ポータルで以下の手順を実行します。

1. [システムエクスプローラ]→[SQL] を選択します。
2. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
3. [クエリ・キャッシュ] フォルダを展開します。
4. そのスキーマの行で [テーブル] リンクを選択します。
5. ページの先頭にある [クエリ・キャッシュ] を選択します。

ポータルには以下のように表示されます。



このリストの各項目は OBJ コードです。

既定では、この OBJ コードよりも前に生成されたルーチンと INT コードは保存されません。そのように生成されたコードも保存することを IRIS に指示できます。“[InterSystems SQL の設定](#)”を参照してください。

クエリ・キャッシュは削除できます（これにより IRIS で強制的にこのコードが再生成されます）。クエリ・キャッシュを削除するには、[アクション]→[クエリキャッシュ削除] を使用します。

10.15 インデックスの構築

IRIS クラスでは、インデックスのメンテナンスは不要です。ただし 1 つの例外として、このクラスのレコードを格納した後でインデックスを追加した場合は、インデックスを構築する必要があります。

そのためには以下の操作を実行します。

1. [システムエクスプローラ]→[SQL] を選択します。
2. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
3. 左側の領域でテーブルを選択します。
4. [アクション]→[インデックス再構築] を選択します。

10.16 テーブルのチューニング機能の使用

特定の SQL クエリを実行する最も効率的な方法をクエリ・オプティマイザが決定するときに、さまざまな要因の中でも特に以下の各項目が考慮されます。

- ・ テーブルにあるレコードの数
- ・ クエリで使用される列の場合、それらの列がどの程度一意であるか

この情報は、特定のテーブルでテーブルのチューニング機能を実行している場合にのみ得られます。この機能はこのデータを計算して、その結果をクラスの `<ExtentSize>` 値および保存されているプロパティの `<Selectivity>` 値として、クラスのストレージ定義と共に保存します。

テーブルのチューニング機能を使用するには以下の手順に従います。

1. [システムエクスプローラ]→[SQL] を選択します。
2. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
3. 左側の領域でテーブルを選択します。
4. [アクション]→[テーブルチューニング] を選択します。

`<Selectivity>` の値については、データの性質が変化していない限り、この処理を再度実行する必要はありません。`<ExtentSize>` の値は正確な値である必要はありません。この値は、さまざまなテーブルのスキャンに要する相対コストを比較するときに使用します。ExtentSize の相対値がテーブルどうしで正確であることが最も重要です (つまり、小さいテーブルの値は小さく、大きいテーブルの値は大きくなる必要があります)。

10.17 1つのデータベースから別のデータベースへのデータの移動

1つのデータベースから別のデータベースへデータを移動する必要がある場合は、以下の手順を実行します。

1. データおよびそのインデックスを収めているグローバルを特定します。
クラスで 사용되는グローバルが不明である場合は、クラスのストレージ定義を確認します。“[ストレージ](#)”を参照してください。
2. これらのグローバルをエクスポートします。そのためには以下の操作を実行します。
 - a. 管理ポータルで、[システムエクスプローラ]→[グローバル] を選択します。
 - b. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
 - c. エクスポートするグローバルを選択します。
 - d. [エクスポート] を選択します。
 - e. グローバルのエクスポート先とするファイルを指定します。絶対パス名または相対パス名によるファイル名をフィールドに入力するか、[参照] を選択して目的のファイルに移動します。
 - f. [エクスポート] を選択します。

.gof を拡張子とするファイルにグローバルがエクスポートされます。

3. これらのグローバルを他のネームスペースにインポートします。そのためには以下の操作を実行します。

- a. 管理ポータルで、[システムエクスプローラ]→[グローバル] を選択します。
 - b. 必要に応じ、ヘッダ領域で [変更] を選択して目的のネームスペースを選択します。
 - c. [インポート] を選択します。
 - d. インポートするファイルを指定します。ファイル名を入力するか、[参照] を選択して目的のファイルに移動します。
 - e. [次へ] を選択して、ファイルの内容を表示します。指定したファイルのグローバルに関する情報のテーブルが表示されます。このテーブルには、各グローバルの名前、それがローカル・ネームスペースまたはデータベースに存在するかどうか、存在する場合は最後に変更されたのはいつかなどの情報が一覧表示されます。
 - f. このテーブルにあるチェック・ボックスを使用して、インポートするグローバルを選択します。
 - g. [インポート] を選択します。
4. “[格納したデータの削除](#)” の説明に従い、最初のデータベースに戻ってグローバルを削除します。

A

Caché の様々な構文

既存の ObjectScript コードを読むと、あまり知られていない構文形式が使用されていることがあります。このページでは、さまざまなグループの構文形式を示し、それが何であるか、その詳細な情報をどこで入手できるのかについて説明します。

このページでは、演算子であることや関数またはコマンドの引数であることが明らかな単一の文字については説明しません。

“ObjectScript リファレンス” の以下の付録も参照してください。

- ・ “ObjectScript で使用する記号” には、すべての記号のリストが記載されています。
- ・ “ObjectScript で使用する省略形” には、コマンドおよび関数の使用可能な短い形式のリストが記載されています。

A.1 “単語” の中の非英数字

このセクションでは、非英数字が中に含まれる単語のような形式のリストを示します。演算子はよく知られているため、それらの多くは明白です。以下はその例です。

`x>5`

あまり明白ではない形式は、以下のとおりです。

`abc^def`

`def` はルーチンであり、`abc` はそのルーチン内のラベルです。`abc^def` はサブルーチンです。

`abc` のバリエーションは、以下のとおりです。

- ・ `%abc`

`def` のバリエーションは、以下のとおりです。

- ・ `%def`
- ・ `def.ghi`
- ・ `%def.ghi`
- ・ `def(xxx)`
- ・ `%def(xxx)`

- ・ `def.ghi(xxx)`
- ・ `%def.ghi(xxx)`

`xxx` は、オプションのコンマ区切りの引数のリストです。

ラベルは、パーセント記号で開始できますが、その後は英数字です。

ルーチン名はパーセント記号で開始でき、1 つ以上のピリオドを含めることができます。キャラットはその名前の一部ではありません。(一般的な使用法では、ルーチンを、その名前の先頭にキャラットが含まれているかのよう参照することはよくあります。したがって、`^def` ルーチンに関するコメントが記述されていることがあります。通常、グローバルを参照しているのか、ルーチンを参照しているのかはコンテキストから判断できます。)

`i%abcdef`

これは、インスタンス変数であり、これを使用してオブジェクトの `abcdef` プロパティの値を取得または設定できます。“クラスの定義と使用”の“[オブジェクト特有の ObjectScript の機能](#)”を参照してください。

この構文は、インスタンス・メソッド内でのみ使用できます。`abcdef` は、同じクラスまたはスーパークラスのプロパティです。

`abc->def`

バリエーション：

- ・ `abc->def->ghi` など

この構文は、InterSystems SQL 文内でのみ使用できます。これは、InterSystems IRIS 矢印構文の例であり、暗黙的な左外部結合を指定します。`abc` は、クエリ対象のクラスのオブジェクト値フィールドであり、`def` は、その子クラスのフィールドです。

`abc->def` は、InterSystems SQL では使用できない InterSystems IRIS ドット構文 (`abc.def`) に類似したものです。

InterSystems IRIS 矢印構文の詳細は、“InterSystems SQL の使用法”の“[暗黙結合 \(矢印構文\)](#)”を参照してください。

`abc?def`

バリエーション：

- ・ `"abc"?def`

疑問符は、パターン・マッチ[演算子](#)です。最初の形式では、この式は、変数 `abc` の値が `def` に指定されたパターンに一致しているかどうかをテストします。2 番目の形式では、`"abc"` はテスト対象の文字列リテラルです。

文字列リテラル `"abc"` と引数 `def` のどちらにも英字以外の文字を含めることができます。

`"abc"["def"]`

バリエーション：

- ・ `abc[def]`
- ・ `abc["def"]`
- ・ `"abc"[def]`

左側の括弧 (`[]`) は、二項包含関係[演算子](#)です。最初の形式では、この式は、文字列リテラル `"abc"` に文字列リテラル `"def"` が含まれるかどうかをテストします。後の形式では、`abc` と `def` が、テスト対象の変数です。

文字列リテラル "abc" と "def" のどちらにも英字以外の文字を含めることができます。

"abc"] "def"

バリエーション :

- ・ `abc]def`
- ・ `abc]"def"`
- ・ `"abc"]def`

右側の括弧 (]) は、二項後続関係演算子です。最初の形式では、この式は、ASCII 文字順で文字列リテラル "abc" が文字列リテラル "def" の後に来るかどうかをテストします。後の形式では、abc と def が、テスト対象の変数です。

文字列リテラル "abc" と "def" のどちらにも英字以外の文字を含めることができます。

"abc"]]"def"

バリエーション :

- ・ `abc]]def`
- ・ `abc]]]"def"`
- ・ `"abc"]]"def`

2 つの連続した右側の括弧 (]]) は、二項前後関係演算子です。最初の形式では、この式は、数値添え字の照合順序で文字列リテラル "abc" が文字列リテラル "abc" の後に順番に並んでいるかどうかをテストします。後の形式では、abc と def が、テスト対象の変数です。

文字列リテラル "abc" と "def" のどちらにも英字以外の文字を含めることができます。

A.2 .(ピリオド 1 つ)

引数リスト内のピリオド

バリエーション :

- ・ `abc.def(.ghi)`
- ・ `abc(.xyz)`

メソッドまたはルーチンを呼び出すと、引数を参照で渡すか、出力として渡すことができます。このためには、引数の前にピリオドを置きます。

行の先頭のピリオド

Do コマンドの命令形式では、数行のコードのグループを 1 つのコード・ブロックにまとめるために、ピリオドの接頭辞を使用します。この従来の Do コマンドは、InterSystems IRIS では使用できません。

A.3 ..(2 つのピリオド)

どの場合も、2 つの連続したピリオドは、クラス・メンバ内から別のクラス・メンバへの参照の始まりです。

..**abcdef**

この構文は、インスタンス・メソッド内でのみ使用できます（ルーチンまたはクラス・メソッド内では使用できません）。**abcdef** は、同じクラスのプロパティです。

..**abcdef (xxx)**

この構文は、メソッド内でのみ使用できます（ルーチン内では使用できません）。**abcdef()** は、同じクラスの別のメソッドであり、**xxx** はオプションの引数のコンマ区切りリストです。

..**#abcdef**

この構文は、メソッド内でのみ使用できます（ルーチン内では使用できません）。**abcdef** は、このクラスのパラメータです。

インターシステムズが提供するクラスでは、慣例ですべてのパラメータがすべて大文字で定義されていますが、作成するコードでは大文字にしなくてもかまいません。

シャープ記号は、パラメータ名の一部ではありません。

A.4 # (シャープ記号)

このセクションでは、シャープ記号で始まる形式のリストを示します。

#abcdef

多くの場合、**#abcdef** はプリプロセッサ指示文です。InterSystems IRIS には、一連のプリプロセッサ指示文が用意されています。これらの名前は、1 つまたは 2 つのシャープ記号で始まります。以下は、いくつかの一般的な例です。

- ・ **#define** は、マクロを定義します（引数を伴うこともあります）。
- ・ **#deflarg** は、コンマを含む 1 つの引数を持つマクロを定義します。
- ・ **#sqlcompile mode** は、後続の埋め込み SQL 文のコンパイル・モードを指定します。

リファレンス情報およびその他の指示文の詳細は、“ObjectScript の使用法”の“[ObjectScript マクロとマクロ・プリプロセッサ](#)”を参照してください。

あまり一般的ではありませんが、形式 **#abcdef** は引数です。この引数は、特定のコマンド ([READ](#) や [WRITE](#) など)、特殊な変数、またはルーチンに使用します。詳細は、この引数を使用するコマンド、変数、またはルーチンのリファレンスの情報を参照してください。

##abcdef

##abcdef はプリプロセッサ指示文です。**#abcdef** の説明を参照してください。

##class(abc.def).ghi(xxx)

バリエーション：

- ・ `##class(def).ghi(xxx)`

`abc.def` はパッケージおよびクラス名であり、`ghi` はそのクラスのクラス・メソッドであり、`xxx` はオプションの引数のコンマ区切りリストです。

パッケージが省略されている場合、クラス `def` はこの参照を含むクラスと同じパッケージ内にあります。

##super()

バリエーション：

- ・ `##super(abcdef)`

この構文は、メソッド内でのみ使用できます。これは、スーパークラスのオーバーライドされたメソッドを、現在のクラス内の同じ名前前の現在のメソッド内から呼び出します。`abcdef` はそのメソッドの引数のコンマ区切りリストです。“クラスの定義と使用”の“[オブジェクト特有の ObjectScript の機能](#)”を参照してください。

A.5 ドル記号 (\$)

このセクションでは、ドル記号で始まる形式のリストを示します。

\$abcdef

通常、`$abcdef` は特殊変数です。“ObjectScript リファレンス”の“[ObjectScript 特殊変数](#)”を参照してください。

`$abcdef` はカスタム特殊変数の場合もあります。“専用のシステム/ツールおよびユーティリティ”の“[%ZLang による ObjectScript の拡張](#)”を参照してください。

\$abcdef(xxx)

通常、`$abcdef()` はシステム関数であり、`xxx` は、オプションの引数のコンマ区切りリストです。詳細は、“[ObjectScript リファレンス](#)”を参照してください。

`$abcdef()` はカスタム関数の場合もあります。“専用のシステム/ツールおよびユーティリティ”の“[%ZLang による ObjectScript の拡張](#)”を参照してください。

\$abc.def.ghi(xxx)

この形式では、`$abc` は `$SYSTEM` (大文字小文字を問わない)、`def` は `%SYSTEM` パッケージのクラスの名前、`ghi` はそのクラスのメソッドの名前、および `xxx` はそのメソッドのオプションの引数のコンマ区切りリストです。

`$SYSTEM` 特殊変数は、`%SYSTEM` パッケージのエイリアスであり、そのパッケージのクラスのメソッドに対して言語に依存しないアクセスを提供します。例えば、`$SYSTEM.SQL.DATEDIFF` のようになります。

このクラスのメソッドの詳細は、[インターシステムズ・クラス・リファレンス](#)を参照してください。

\$\$abc

バリエーション：

- ・ `$$abc(xxx)`

abc は、この参照を含むルーチンまたはメソッド内で定義されたサブルーチンです。この構文は、サブルーチン abc を呼び出し、その返り値を取得します。“ObjectScript の使用法”の“[ユーザ定義コード](#)”を参照してください。

\$\$abc^def

バリエーション：

- ・ `$$abc^def (xxx)`
- ・ `$$abc^def .ghi`
- ・ `$$abc^def .ghi (xxx)`

この構文は、サブルーチン abc を呼び出し、その返り値を取得します。キャレットの後の部分は、このサブルーチンを含むルーチンの名前です。“ObjectScript の使用法”の“[ユーザ定義コード](#)”を参照してください。

\$\$\$abcdef

abcdef はマクロです。ドル記号はその名前の一部ではありません (したがって、マクロ定義には表示されません)。

InterSystems IRIS に用意されているマクロのいくつかは、“ObjectScript の使用法”の“[システムにより提供されるマクロのリファレンス](#)”に記載されています。

一般的な使用法では、マクロを、その名前にドル記号が含まれているかのように参照することはよくあります。したがって、\$\$\$abcdef マクロに関するコメントが記述されていることがあります。

A.6 パーセント記号 (%)

規約では、InterSystems IRIS システム・クラスの大部分のパッケージ、クラス、およびメソッドは、パーセント文字で始まります。調べている要素がこれらの 1 つであるかどうかはコンテキストから明確になります。それ以外の場合は、以下の可能性があります。

%abcdef

%abcdef は、以下のいずれかです。

- ・ ローカル変数。InterSystems IRIS によって設定されるローカル変数も含みます。
- ・ ルーチン。

バリエーション：

- `%abcdef.ghijkl`

- ・ 埋め込み SQL 変数 (それらは %msg、%ok、%ROWCOUNT、および %ROWID です)。

詳細は、“InterSystems SQL の使用法”の“[埋め込み SQL のホスト変数](#)”を参照してください。

- ・ InterSystems SQL コマンド、関数、または述語条件 (例えば、%STARTSWITH および %SQLUPPER)。

バリエーション：

- `%abcdef (xxx)`

詳細は、“[InterSystems SQL リファレンス](#)”を参照してください。

%%abcdef

%%abcdef は %%CLASSNAME、%%CLASSNAMEQ、%%ID、または %%TABLENAME です。これらは、擬似フィールド・キーワードです。詳細は、“[InterSystems SQL リファレンス](#)”を参照してください。

A.7 キャレット (^)

このセクションでは、キャレットで始まる形式を、よく使われるものから順に示します。

^abcdef

バリエーション：

- ・ ^%abcdef

これには以下の 3 つの可能性があります。

- ・ ^abcdef または ^%abcdef はグローバルです。
- ・ ^abcdef または ^%abcdef は、LOCK コマンドの引数です。この場合、^abcdef または ^%abcdef は、ロック名になり、ロック・テーブル (メモリ内) に保持されます。
- ・ abcdef または %abcdef はルーチンです。キャレットは名前の一部ではなく、ルーチンを呼び出す構文の一部です。

一般的な使用方法では、ルーチンの名前の先頭にキャレットが含まれているものとしてルーチンを参照することはよくあります。したがって、^abcdef ルーチンに関するコメントが記述されていることがあります。通常、グローバルを参照しているのか、ルーチンを参照しているのかはコンテキストから判断できます。ロック名は、LOCK コマンドの後にのみ指定されます。その他のコンテキストで、ロック名を使用することはできません。

^\$abcdef

バリエーション：

- ・ ^\$|"ghijkl"|abcdef

これらはそれぞれ、構造化システム変数であり、グローバル、ジョブ、ロック、またはルーチンに関する情報を提供します。

\$abcdef は \$GLOBAL、\$JOB、\$LOCK、または \$ROUTINE です。

ghijkl はネームスペース名です。

InterSystems IRIS では、以下のシステム変数に情報を格納します。

- ・ ^\$GLOBAL
- ・ ^\$JOB
- ・ ^\$LOCK
- ・ ^\$ROUTINE

“ObjectScript リファレンス”を参照してください。

^||abcdef

バリエーション：

- ・ `^| "^"|abcdef`
- ・ `^["^"]abcdef`
- ・ `^["^", ""]abcdef`

これらはそれぞれ、サイズの大きいデータ値を一時的に格納するためのメカニズムであるプロセス・プライベート・グローバルです。InterSystems IRIS では、いくつかが内部で使用されていますが、パブリックに使用するために提供されていません。ユーザ独自のプロセス・プライベート・グローバルを定義し、使用できます。“ObjectScript の使用法” の “[変数](#)” を参照してください。

`^|xxx|abcdef`

いくつかのバリエーション：

- ・ `^|xxx|%abcdef`
- ・ `^[xxx]abcdef`
- ・ `^[xxx]%abcdef`

これらはそれぞれ、拡張参照 (別のネームスペース内のグローバルまたはルーチンの参照) です。以下の可能性があります。

- ・ `^abcdef` または `^%abcdef` は、他のネームスペースのグローバルです。
- ・ `abcdef` または `%abcdef` は、他のネームスペースのグローバルです。

xxx コンポーネントはネームスペースを示します。これは、引用符で囲んだ文字列または引用符で囲まれていない文字列です。“ObjectScript の使用法” の “[構文規則](#)” の “[拡張参照](#)” を参照してください。

`^abc^def`

これは暗黙のネームスペースです。“ObjectScript リファレンス” の “[ZNSPACE](#)” を参照してください。

`^^abcdef`

これは暗黙のネームスペースです。“ObjectScript リファレンス” の [ZNSPACE](#) のエントリを参照してください。

A.8 その他の形式

`+abcdef`

いくつかのバリエーション：

- ・ `^+abcdef`
- ・ `+ "abcdef"`

これらの式はそれぞれ、数値を返します。最初のバージョンでは、`abcdef` はローカル変数の名前です。この変数の内容が数値文字で始まらない場合、この式は 0 を返します。内容が数値文字で始まる場合、この式はその数値文字と、それ以降のすべての数値文字を (数値以外の文字が初めて出現するまで) 返します。ここでは、以下の例を実行します。

ObjectScript

```
write +"123abc456"
```

“ObjectScript の使用法” の “[文字列関係演算子](#)” を参照してください。

{ "abc": (def), "abc": (def), "abc": (def) }

この構文は [JSON オブジェクト・リテラル](#) であり、%DynamicObject インスタンスを返します。"abc" はプロパティの名前、def はプロパティの値です。詳細は、"[JSON の使用](#)" を参照してください。

{abcdef}

この構文は、InterSystems SQL で ObjectScript が使用されている場合に使用されることがあります。abcdef はフィールドの名前です。“クラスの定義と使用” の “[ObjectScript からのフィールドの参照](#)” を参照してください。

{%%CLASSNAME}

この構文は、トリガ・コード内で使用でき、クラスのコンパイル時に置換されます。

その他：

- ・ {%%CLASSNAMEQ}
- ・ {%%ID}
- ・ {%%TABLENAME}

これらの項目では、大文字と小文字は区別されません。“InterSystems SQL リファレンス” の “[CREATE TRIGGER](#)” を参照してください。

&sql (xxx)

これは埋め込み SQL であり、ObjectScript が使用されている場所であればどこでも使用できます。xxx は 1 つの SQL 文です。“InterSystems SQL の使用法” の “[埋め込み SQL の使用法](#)” を参照してください。

[abcdef,abcdef,abcdef]

この構文は [JSON 配列リテラル](#) であり、%DynamicArray インスタンスを返します。abcdef は配列の項目です。詳細は、“[JSON の使用](#)” を参照してください。

***abcdef**

以下の関数およびコマンドによって使用される特別な構文です。

- ・ [\\$ZSEARCH](#)
- ・ [\\$EXTRACT](#)
- ・ [WRITE](#)
- ・ [\\$ZTRAP](#)
- ・ [\\$ZERROR](#)

“ObjectScript リファレンス” を参照してください。

?abcdef

疑問符は、パターン・マッチ[演算子](#)であり、abcdef は比較パターンです。

@abcdef

アット記号は、間接[演算子](#)です。

B

識別子のルールとガイドライン

使用上の便宜のために、このページでは、すべてのサーバ側コンテキストにおける ObjectScript 識別子のルールを要約し、名前衝突を回避するためのガイドラインを提供します。

ObjectScript には、予約語がありません。そのため、識別子としてコマンドを使用しても結果は構文的に正しくなりますが、そのコードはそれを読む人にわかりにくくなる可能性があります。

ユーザ、ロール、リソースなどのセキュリティ・エンティティの識別子については、“[承認ガイド](#)”の関連するセクションを参照してください。

B.1 ネームスペース

ネームスペース名では、最初の文字は、英字またはパーセント記号(%)に限られます。残りの文字は、英字、数字、ハイフン、またはアンダースコアにする必要があります。この名前は、255 文字以内にする必要があります。

B.1.1 回避する必要があるネームスペース名

予約済みのネームスペース名は、%SYS、BIN、BROKER、および DOCUMENTIC です。これらのネームスペースのいくつかについては、このドキュメントで前述した“[システム提供ネームスペース](#)”で説明しています。他のものは InterSystems IRIS によって内部で使用されます。

B.2 データベース

データベース名では、最初の文字は、英字またはアンダースコアに限られます。残りの文字は、英字、数字、ハイフン、またはアンダースコアにする必要があります。この名前は、30 文字以内にする必要があります。

B.2.1 回避する必要があるデータベース名

予約済みのデータベース名は、IRISLOCALDATA、IRISSYS、IRISLIB、IRISAUDIT、IRISTEMP です。

これらのデータベースの詳細は、このドキュメント内の前述の“[システム提供データベース](#)”を参照してください。

B.3 ローカル変数

ローカル変数の名前について、ObjectScript では以下のルールが適用されます。

- ・ 最初の文字は、英字かパーセント記号 (%) に限られます。
% で始まる名前にする場合、その直後の文字は `z` または `Z` にします。
- ・ 残りの文字は、英字または数字にする必要があり、ASCII 128 より大きい任意の文字も使用できます。
- ・ 名前は、大文字と小文字を区別します。
- ・ 名前は、最初の 31 文字で (該当するコンテキストで) 一意である必要があります。
変数の添え字は、このカウントに含まれません。

B.3.1 回避する必要があるローカル変数名

ローカル変数に対して以下の名前は使用しないでください。

- ・ `SQLCODE`

InterSystems SQL が実行される可能性があるコンテキストでは、変数の名前として `SQLCODE` を使用しないでください。"InterSystems IRIS エラー・リファレンス" の "[SQLCODE 値とエラー・メッセージ](#)" を参照してください。

B.4 グローバル変数

グローバル変数の名前について、ObjectScript では以下のルールが適用されます。

- ・ 最初の文字はキャレット (^) にし、次の文字は英字かパーセント記号 (%) にする必要があります。グローバル名の場合、文字は ASCII 65 から ASCII 255 の範囲のアルファベット文字として定義されます。
- ・ 残りの文字は、英字または数字にする必要があります (ただし、次の箇条書きに示すように例外が 1 つあります)。
- ・ グローバル変数の名前には、1 つ以上のピリオド (.) 文字を含めることができますが、最初と最後の文字には使用できません。
- ・ 名前は、大文字と小文字を区別します。
- ・ 名前は、最初の 31 文字が (該当するコンテキストで) 一意である必要があります。キャレット文字はこのカウントに含まれません。つまり、グローバル変数の名前は、キャレットを含めて最初の 32 文字が一意である必要があります。
変数の添え字は、このカウントに含まれません。
- ・ **IRISSYS** データベースでは、グローバル名はすべて予約されていますが、先頭に `^z`、`^Z`、`^%z`、`^%Z` が付くものについては例外です。"[IRISSYS データベースおよびカスタム項目](#)" を参照してください。

他のすべてのデータベースでは、先頭に `^IRIS.` および `^%IRIS.` が付くグローバル名はすべて予約されています。

[以下のサブセクション](#)も参照してください。

B.4.1 回避する必要があるグローバル変数名

データベースを作成すると、InterSystems IRIS によって、それ自体が使用するためのいくつかのグローバルでそれが初期化されます。また、作成するネームスペースすべてに、システム・グローバルへのマッピングが含まれます。これには、書き込み可能システム・データベース内のグローバル・ノードも含まれます。

システム・グローバルを上書きしないようにするために、ネームスペースでは以下のグローバルを設定、変更、または削除しないでください。

- ・ 以下の例外を除く、`%` で始まる名前を持つグローバル
 - `%z` または `%Z` で始まる名前を持つ独自のグローバル
 - `%SYS` (ドキュメントに記載されているようにノードを設定できます)
- ・ `BfVY` (InterSystems IRIS の[シャーディング](#)で使用するために予約されています)
- ・ `CacheTemp*` (InterSystems IRIS の一部のバージョンで使用するために予約されています)
- ・ `DeepSee.*` (制限は、InterSystems IRIS Analytics を使用しているネームスペースにのみ適用されます)
- ・ `Ens*` (制限は、相互運用対応ネームスペースにのみ適用されます。“相互運用プロダクションの概要”を参照してください。)
- ・ `ERRORS`
- ・ `InterSystems.Sequences` (制限は、InterSystems IRIS Hibernate Dialect を使用しているネームスペースにのみ適用されます)
- ・ `IRIS.*` (インターシステムズで使用するために予約されています)
- ・ `IRISAuditD`
- ・ `IS` (InterSystems IRIS の[シャーディング](#)で使用するために予約されています)
- ・ `ISDebugLevel` (ドキュメントに記載されているようにノードを設定する場合を除く)
- ・ `ISCMonitor` (ドキュメントに記載されているようにノードを設定する場合を除く)
- ・ `ISCSOAP` (ドキュメントに記載されているようにノードを設定する場合を除く)
- ・ `ISCMethodWhitelist`
- ・ `mqh` (SQL クエリの履歴)
- ・ `mtemp*`
- ・ `OBJ.GUID` (ドキュメントに記載されている場合を除く)
- ・ `OBJ.DSTIME`
- ・ `OBJ.JournalT`
- ・ `oddBIND`
- ・ `oddCOM`
- ・ `oddDEF` (クラス定義を含む)
- ・ `oddDEP`
- ・ `oddEXT`
- ・ `oddEXTR`
- ・ `oddMAP`

- ・ `^oddMETA`
- ・ `^oddPKG`
- ・ `^oddPROC`
- ・ `^oddPROJECT`
- ・ `^oddSQL`
- ・ `^oddStudioDocument`
- ・ `^oddStudioMenu`
- ・ `^oddTSQL`
- ・ `^oddXML`
- ・ `^rBACKUP`
- ・ `^rINC` (インクルード・ファイルを含む)
- ・ `^rINCSAVE`
- ・ `^rINDEX`
- ・ `^rINDEXCLASS`
- ・ `^rINDEXEXT`
- ・ `^rINDEXSQL`
- ・ `^rMAC` (MAC コードを含む)
- ・ `^rMACSAVE`
- ・ `^rMAP`
- ・ `^rOBJ` (OBJ コードを格納する)
- ・ `^ROUTINE` (ルーチンを格納する)
- ・ `^SPOOL` (制限は、InterSystems IRIS スプーリングを使用しているネームスペースにのみ適用されます。“入出力デバイス・ガイド”の“[スプール・デバイス](#)”を参照してください。)
- ・ `^SYS` (ドキュメントに記載されているようにノードを設定する場合を除く)
- ・ `^z*` および `^Z*` (インターシステムズで使用するために予約されています)

B.5 ルーチンとラベル

ルーチンまたはラベルの名前について、ObjectScript では以下のルールが適用されます。

- ・ 最初の文字は、英字かパーセント記号 (%) に限られます。
% で始まるルーチン名にする場合、その直後の文字は `z` または `Z` にします。“[IRISSYS データベースおよびカスタム項目](#)”を参照してください。
- ・ 残りの文字は、英字または数字にする必要があります (ただし、例外が 1 つあります。次の箇条書きを参照してください)。これらの他の文字に、ASCII 128 より大きい任意の文字を使用できます。
- ・ ルーチンの名前には、1 つ以上のピリオド (.) 文字を含めることができますが、最初と最後の文字には使用できません。

- ・ 名前は、大文字と小文字を区別します。
- ・ ルーチンの名前は、最初の 255 文字以内で一意（該当するコンテキスト内）である必要があります。
ラベルは、最初の 31 文字以内で一意（該当するコンテキスト内）である必要があります。

特定の %Z ルーチン名は、ユーザが使用するために予約されています。“[IRISSYS データベースおよびカスタム項目](#)”を参照してください。

B.5.1 ユーザが使用するために予約されているルーチン名

InterSystems IRIS では、ユーザが使用するために、以下のルーチン名が予約されています。これらのルーチンは存在しませんが、それらを定義した場合、これらのイベントが発生したときにそれらがシステムによって自動的に呼び出されます。

- ・ ^ZWELCOME ルーチンは、ターミナルが起動したときに実行するカスタム・コードを含めることを目的としています。“[ターミナルの使用法](#)”を参照してください。
- ・ ^ZAUTHENTICATE および ^ZAUTHORIZE ルーチンは、認証および承認のためのカスタム・コードを含めることを目的としています（代行認証と代行承認をサポートするため）。このようなルーチンのために、InterSystems IRIS にはテンプレートが用意されています。“[代行承認の使用法](#)”と“[代行認証](#)”を参照してください。
- ・ ^ZMIRROR ルーチンは、InterSystems IRIS ミラーリングを使用する場合の、フェイルオーバー動作をカスタマイズするためのコードを含めることを目的としています。“[高可用性ガイド](#)”を参照してください。
- ・ ^%ZSTART および ^%ZSTOP ルーチンは、ユーザ・ログインなどの特定のイベントが発生したときに実行されるカスタム・コードを含めることを目的としています。それらを定義した場合、これらのイベントが発生したときにそれらがシステムによって呼び出されます。“[専用のシステム/ツールおよびユーティリティ](#)”の“[%^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](#)”を参照してください。
- ・ %ZLANGV00、および名前が %ZLANG で始まる他のルーチンは、カスタム変数、コマンド、および関数を含めることを目的としています。“[専用のシステム/ツールおよびユーティリティ](#)”の“[%^ZLANG ルーチンによる言語の拡張](#)”を参照してください。
- ・ ^%ZJREAD ルーチンは、^JCONVERT ルーチンを使用する場合にジャーナル・ファイルを操作するためのロジックを含めることを目的としています。“[データ整合性ガイド](#)”の“[ジャーナリング](#)”を参照してください。

B.6 クラス

どのクラスも、完全クラス名は `packagename.classname` の形式です。

クラス名のルールは、以下のとおりです。

- ・ `packagename`（パッケージ名）および `classname`（短いクラス名）は英字で始まる必要があります。
- ・ `packagename` にはピリオドを含めることができます。
その場合、ピリオドの直後の文字は英字にする必要があります。
`packagename` のピリオドで区切られた各部は、サブパッケージ名として扱われ、一意性のルールに従います。
- ・ 残りの文字は、英字または数字にする必要があります、ASCII 128 より大きい文字も使用できます。
- ・ パッケージ名と短いクラス名は一意である必要があります。同様に、サブパッケージ名は、親パッケージ名内で一意である必要があります。

各クラスを定義したときに使用した大文字と小文字はシステムで保持され、クラス定義に指定した大文字と小文字に完全に一致させる必要があります。ただし、大文字と小文字のみが違う2つの識別子を指定することはできません。例えば、識別子 `id1` と `ID1` は一意性を保つ目的からは同一と見なされます。

- ・ 相互運用対応ネームスペースでは、**Ens**、**EnsLib**、**EnsPortal**、または **CSPX** をパッケージ名として使用しないでください。これらのパッケージは、アップグレード・プロセス中に完全に置換されます。これらのパッケージ内でクラスを定義した場合は、アップグレード前にそれらのクラスをエクスポートして、アップグレード後にインポートする必要があります。
- ・ いずれの相互運用対応ネームスペースでも、先頭に **Ens** (大文字と小文字の区別あり) が付くパッケージ名を使用しないことをお勧めします。詳細は、“プロダクションの開発の概要”の“環境上の考慮事項”を参照してください。
- ・ 以下のように長さの制限があります。
 - パッケージ名 (すべてのピリオドを含む) は最初の 189 文字内で一意である必要があります。
 - 短いクラス名は最初の 60 文字内で一意である必要があります。

完全なクラス名は、クラス・メンバの個別の長さ制限に寄与します。次のセクションを参照してください。

B.6.1 回避する必要があるクラス名

永続クラスでは、クラスの短い名前として SQL 予約語を使用しないでください。

クラスの短い名前として SQL 予約語を使用する場合、そのクラスに対して `SqlTableName` キーワードを指定することが必要になります。また、短いクラス名と SQL テーブル名の間に不一致があると、将来コードを読むときに注意が必要になります。

B.7 クラス・メンバ

作成するクラス・メンバについては、その項目の名前が範囲指定されていない限り、以下のルールに従う必要があります。

- ・ 名前は、英字かパーセント記号 (%) で始まる必要があります。

SQL に投影されるクラス・メンバについては、その他の考慮事項があります (例えば、このメンバには永続クラスのほとんどのプロパティが含まれます)。最初の文字が % である場合、2 番目の文字は `z` または `z` である必要があります。
- ・ 残りの文字は、英字または数字にする必要があり、ASCII 128 より大きい文字も使用できます。
- ・ メンバ名は (該当するコンテキスト内で) 一意である必要があります。

クラスを定義したときに使用した大文字と小文字はシステムで保持され、クラス定義に指定した大文字と小文字に完全に一致させる必要があります。ただし、2 つのクラス・メンバに、大文字と小文字のみが違う名前を指定することはできません。例えば、識別子 `id1` と `ID1` は一意性を保つ目的からは同一と見なされます。
- ・ メソッドまたはプロパティ名は最初の 180 文字内で一意である必要があります。
- ・ プロパティの名前の長さ、そのプロパティのインデックスの名前の長さを合わせた長さは、180 文字を超えてはいけません。
- ・ 各メンバの完全な名前 (未修飾のメンバ名および完全なクラス名を含む) は、220 文字以下である必要があります。
- ・ 2 つのメンバに同一の名前を付与しないでください。予測できない結果となる可能性があります。

メンバ名の範囲を指定することもできます。範囲指定したメンバ名を作成するには、名前の最初と最後の文字に二重引用符を使用します。これにより、範囲指定しない場合には許可されない文字を名前に含めることができます。以下はその例です。

Class Member

```
Property "My Property" As %String;
```

B.7.1 回避する必要があるメンバ名

永続クラスについては、メンバの名前として SQL 予約語を使用しないでください。

これらの名前の 1 つに SQL 予約語を使用すると、そのクラスを SQL に投影する方法を指定するときに余分な作業が必要になります。例えば、プロパティの場合、`SqlFieldName` キーワードを指定する必要があります。また、クラスの識別子と SQL の識別子の間に不一致があると、将来コードを読むときに注意が必要になります。

SQL 予約語には、`%SQLUPPER` や `%FIND` のように、名前が `%` で始まる項目が数多く存在する点に注目してください。このような項目は、SQL に対する InterSystems 拡張機能です。将来のリリースで、その他の拡張機能が追加される可能性もあります。

B.8 IRISSYS データベースおよびカスタム項目

IRISSYS データベースに項目を作成できます。ただし、以下の命名規約に従っていない場合、InterSystems IRIS のアップグレード・プロセスで IRISSYS 内のカスタム項目は削除されます。

項目が上書きされないようにこのデータベースにコードまたはデータを追加するには、以下のいずれかを実行します。

- ・ `%SYS` ネームスペースに移動し、項目を作成します。このネームスペースの場合、既定のルーチン・データベースおよび既定のグローバル・データベースは、どちらも IRISSYS です。以下の名前付け規約を使用して、項目がアップグレード・インストールの影響を受けないようにします。
 - クラス：パッケージを `Z` または `z` で始めます。
 - ルーチン：名前を `Z`、`z`、`%Z`、または `%z` で始めます。
 - グローバル：名前を `^Z`、`^z`、`^%Z`、または `^%z` で始めます。
- ・ どのネームスペースでも、以下の名前で項目を作成します。
 - ルーチン：名前を `%Z` または `%z` で始めます。
 - グローバル：名前を `^%Z` または `^%z` で始めます。

ネームスペースの標準マッピングにより、これらの項目は IRISSYS に書き込まれます。

MAC コードとインクルード・ファイルはアップグレード・プロセスによる影響を受けません。

別の方法で項目をすべてのネームスペースからアクセス可能にするには、`%ALL` ネームスペースを使用します。これは、“すべてのネームスペースへのデータのマッピング” で説明しています。

C

一般的なシステム制限

このページでは、InterSystems IRIS® のシステム制限のいくつかを示します。

識別子の名前の制限については、“[識別子のルールとガイドライン](#)” を参照してください。

その他のシステム全体の制限は、“[構成パラメータ・ファイル・リファレンス](#)” を参照してください。

C.1 文字列長の制限

文字列の値の長さには、3,641,144 文字という制限があります。

重要 “文字列” は、入出力デバイスからの読み取りの結果に限られないことを理解することが重要です。その他のコンテキストでも、文字列は出現します。例えば、結果セット (SQL クエリ、多数の項目を含む \$LIST の構築、または XSLT 変換などの出力として返される) の行に含まれるデータのコンテキストにも文字列が出現します。

C.2 クラスの制限

以下の制限はクラスにのみ適用されます。

クラス継承階層

制限：50。1 つのクラスは最大 50 階層までのサブクラスを作成できます。

外部キー

制限：1 クラスあたり 400。

インデックス

制限：1 クラスあたり 400。

メソッド

制限：1 クラスあたり 2000。

パラメータ

制限：1 クラスあたり 1000。

プロジェクション

制限：1 クラスあたり 200。

プロパティ

制限：1 クラスあたり 1000。

クエリ

制限：1 クラスあたり 200。

SQL 制約

制限：1 クラスあたり 200。

ストレージ定義

制限：1 クラスあたり 10。

スーパークラス

制限：1 クラスあたり 127。

トリガ

制限：1 クラスあたり 200。

XData ブロック

制限：1 クラスあたり 1000。

C.3 クラスおよびルーチンの制限

以下の制限はクラスとルーチンに適用されます。

クラス・メソッド参照

制限：1 ルーチンまたは 1 クラスあたり 32768 の一意の参照。

以下は、メソッド名が同じでもクラス名が異なるため、2 つのクラス・メソッド参照としてカウントされます。

ObjectScript

```
Do ##class(c1).abc(), ##class(c2).abc()
```

クラス名参照

制限：1 ルーチンまたは 1 クラスあたり 32768 の一意の参照。

例えば、以下は 2 つのクラス名参照としてカウントされます。

```
Do ##class(c1).abc(), ##class(c2).abc()
```

同様に、以下も、2 つのクラス参照としてカウントされます。これは、%File から %Library.File への正規化がコンパイル時ではなく実行時に行われるためです。

```
Do ##class(%File).Open(x)
Do ##class(%Library.File).Open(y)
```

インスタンス・メソッド参照

制限：1 ルーチンまたは 1 クラスあたり 32768。

X と Y が OREF の場合、以下は、1 つのインスタンス・メソッド参照としてカウントされます。

```
Do X.abc(), Y.abc()
```

多次元プロパティへの参照は、インスタンス・メソッドとしてカウントされます。これはコンパイラがこれらを区別できないためです。例えば、以下の文を考えてみます。

```
Set var = OREF.xyz(3)
```

コンパイラは、この文が xyz() メソッドを参照しているか、または多次元プロパティ xyz を参照しているかを区別できません。そのため、コンパイラはこれを、インスタンス・メソッド参照としてカウントします。

行数

制限：1 ルーチンあたり 65535 行（コメント行を含む）。この制限は、INT 表現のサイズに適用されます。

リテラル (ASCII)

制限：1 ルーチンまたは 1 クラスあたり 65535 の ASCII リテラル。

ASCII リテラルは、どの文字も \$CHAR(255) を超えない 3 つ以上の文字を含む引用符付き文字列です。

ASCII リテラルと Unicode リテラルは別々に処理され、個別の制限があります。

リテラル (Unicode)

制限：1 ルーチンまたは 1 クラスあたり 65535 の Unicode リテラル。

Unicode リテラルは、1 つ以上の文字が \$CHAR(255) を超える引用符付き文字列です。

ASCII リテラルと Unicode リテラルは別々に処理され、個別の制限があります。

パラメータ数

制限：サブルーチン、メソッド、またはスタッド・プロシージャあたり 255 のパラメータ。

プロシージャ数

制限：1 ルーチンあたり 32767。

プロパティ読み取り参照

制限：1 ルーチンまたは 1 クラスあたり 32768。

この制限は、以下の例のように、プロパティの値の読み取りに関するものです。

ObjectScript

```
Set X = OREF.prop
```

プロパティ設定参照

制限：1 ルーチンまたは 1 クラスあたり 32768。

この制限は、以下の例のように、プロパティの値の設定に関するものです。

ObjectScript

```
Set OREF.prop = value
```

ルーチン参照

制限：1 ルーチンまたは 1 クラスあたり 65535。

この制限は、ルーチンまたはクラスの一意の参照（routine）の数に適用されます。

ターゲット参照

制限：1 ルーチンまたは 1 クラスあたり 65535。

ターゲットは label^routine（ラベルとルーチンの組み合わせ）です。

ターゲット参照はルーチン参照としてもカウントされます。例えば、以下は 2 つのルーチン参照と 3 つのターゲット参照としてカウントされます。

```
Do Label1^Rtn, Label2^Rtn, Label1^Rtn2
```

TRY ブロック数

制限：1 ルーチンあたり 65535。

変数（プライベート）

制限（ObjectScript）：1 プロシージャあたり 32763。

変数（パブリック）

制限（ObjectScript）：1 ルーチンまたは 1 クラスあたり 65503。

変数名やその他の識別子の長さ制限の詳細は、“[識別子のルールとガイドライン](#)”を参照してください。

C.4 その他のプログラミング制限

以下のテーブルに、コードを記述する際に関連するその他の制限を示します。

%Status 値の制限

エラー・メッセージの長さ制限：32000 文字未満。

単一の %Status 値に結合できる %Status 値の最大数：150。

{ } による入れ子

制限：32767 レベル。

これは、中括弧を使用するあらゆる言語要素の入れ子の最大の深さです。例：IF { FOR { WHILE {...}}

1 行あたりの文字数

制限：1 行あたり 65535 文字。

グローバル添え字、長さ

“グローバルの使用法”の“[グローバル参照の最大長](#)”を参照してください。

グローバル参照、長さ

制限：511 文字 (エンコードされた文字数、入力した文字数 511 より少なくなる場合があります)。グローバル参照という用語は、グローバルの名前と、その添え字をすべて合わせたものを指します。詳細は、“グローバルの使用法” の [“グローバル参照の最大長”](#) を参照してください。

数値

制限 (10 進数形式またはネイティブ形式)：約 $1.0\text{E}-128$ ～ $9.22\text{E}145$ 。“[インターシステムズ・アプリケーションでの数値の計算](#)” を参照してください。

制限 (倍精度形式)：“[インターシステムズ・アプリケーションでの数値の計算](#)” を参照してください。

D

インターシステムズ・アプリケーションでの数値の計算

このページでは、InterSystems IRIS® によってサポートされている数値形式について、詳しく説明します。

D.1 数値の表現

InterSystems IRIS で数値を表現する方法には 2 種類あります。

- ・ 1 つは InterSystems IRIS オリジナルの実装当初からあったもので、10 進形式と呼ばれます。
クラス定義では、プロパティに小数点形式の数を含めるときに、`%Library.Decimal` データ型クラスを使用します。
- ・ もう 1 つは IEEE Binary Floating-Point Arithmetic 標準 ([#754-1985](#)) に準拠した形式で、近年サポートされるようになりました。この後者の形式は `$DOUBLE` 形式と呼ばれ、数値をこの形式に変換するために使用される ObjectScript 関数 (`$DOUBLE`) にちなんで名付けられたものです。
クラス定義では、プロパティに `$DOUBLE` 形式の数を含めるときに、`%Library.Double` データ型クラスを使用します。

D.1.1 10 進形式

InterSystems IRIS では、10 進数は内部的に 2 つの部分で表されます。最初の部分は仮数、2 番目の部分は指数と呼ばれます。

- ・ 仮数は、対象となる値の有効桁数で構成されます。符号付 64 ビット整数として保存され、その値の右側には小数点があると見なされます。精度を失うことなく表現できる指数 0 の最大の正の整数は 9,223,372,036,854,775,807、最大の負の整数は -9,223,372,036,854,775,808 です。
- ・ 指数は、符号付バイトとして内部的に保存されます。値の範囲は 127 ~ -128 です。
これは、値の 10 進数の指数です。つまり、その数の値は、仮数に 10 の指数のべき乗を乗算したものになります。

例えば、ObjectScript リテラル値 1.23 の場合、仮数は 123 であり、指数は -2 です。

したがって、InterSystems IRIS がネイティブ形式で表現できる数値の範囲は、約 $1.0\text{E}-128 \sim 9.22\text{E}145$ になります。(最初の値は最小の指数を持つ最小の整数になります。2 番目の値は最も大きい整数で、小数点が左に移動して、それに従って指数が増加する形式で表されます。)

小数桁数が 18 の数値はすべて正確に表現できます。また、仮数が表現範囲内にある数値は、19 桁の値として正確に表現できます。

注釈 InterSystems IRIS では、数値を 10 進形式にする必要がない限り、仮数を正規化することはありません。したがって、仮数が 123 で指数が 1 の数と、仮数が 1230 で指数が 0 の数は同じものを表します。

D.1.2 \$DOUBLE 形式

インターシステムズの \$DOUBLE 形式は、[IEEE-754-1985](#)、具体的には 64 ビットのバイナリ (倍精度) 表現に準拠します。つまり、以下の 3 つの部分で構成されます。

- ・ 符号ビット
- ・ 2 つの指数 の 11 ビット乗。指数値には 1023 のバイアス分が含まれるので、\$DOUBLE(1.0) の指数の内部値は 0 ではなく 1023 になります。
- ・ 正の 52 ビット小数の仮数部仮数部は常に正の数値として処理され正規化されるため、仮数部として表されていないくても、1 ビットは先頭のバイナリ桁であると見なされます。したがって、仮数部は数値としては 53 ビット長になり、値 1 の後に暗黙の 2 進小数点、その後には小数の仮数部が続きます。これは、暗黙的に 2^{*52} で除算された整数として考えることができます。

整数として、0 ~ 9,007,199,254,740,992 のすべての値は正しく表現できます。大きな整数を正しく表現できるかどうかは、そのビット・パターンによります。

このデータ表現には、InterSystems IRIS のネイティブ形式にはない以下の 3 つのオプション機能があります。

- ・ (負の数の平方根を取るなど) 不正な計算結果を NaN (非数) として表現する機能。
- ・ +0 および -0 の両方を表現する機能。
- ・ 無限大を表現する機能。
- ・ この標準は、 2^{*-1022} より小さい数の表現を提供します。これは、“緩やかな精度の損失” と呼ばれる手法で行われます。詳細は、[標準](#)を参照してください。

これらの機能は、個別プロセスの場合は `%SYSTEM.Process` クラスの `IEEEError()` メソッド、システム全般の場合は `Config.Miscellaneous` クラスの `IEEEError()` メソッドを介してプログラム制御されます。

重要 IEEE バイナリ浮動小数点表現を使用して計算すると、同じ IEEE 演算でも異なる結果が得られる場合があります。インターシステムズでは、以下に対して独自の実装を記述しています。

1. \$DOUBLE バイナリ浮動小数点と 10 進数との変換。
2. \$DOUBLE と数値文字列との変換。
3. \$DOUBLE とその他の数値型と間の比較。

これにより、\$DOUBLE 値を InterSystems IRIS データベースに挿入したり、InterSystems IRIS データベースからフェッチしたりするときに、すべてのハードウェア・プラットフォームで結果が同じになることが保証されます。

ただし、\$DOUBLE 型を含むその他すべての計算では、InterSystems IRIS はベンダが提供する浮動小数点ライブラリのサブルーチンを使用します。そのため、同じ演算のセットでも、プラットフォームによってわずかな差異が生じる可能性があります。ただし、すべてのケースにおいて、インターシステムズの \$DOUBLE の計算は、C の double 型で実行するローカルの計算と等しくなります。つまり、インターシステムズの \$DOUBLE の計算に対するプラットフォーム間の差異は、最大でも、それぞれ同じプラットフォーム上で実行される C プログラムの IEEE 値の計算結果の差異に留まります。

D.1.3 SQL 表現

InterSystems SQL データ型の DOUBLE と DOUBLE PRECISION は、IEEE 浮動小数点数 (つまり \$DOUBLE) を表します。SQL FLOAT データ型は、標準の InterSystems IRIS の 10 進数を表します。

D.2 数値形式の選択

使用する形式の選択は、計算の要件に大きく依存します。InterSystems IRIS の 10 進形式では 18 桁を超える精度が許可されますが、\$DOUBLE では 15桁しか保証されません。

多くの場合、10 進形式は使用法がより簡単で、結果も正確です。予想どおりの結果が返されるため、通常、(通貨計算などの) 10 進値の計算に使用されます。ほとんどの 10 進数の小数は、バイナリ的小数ほど正確には表現できません。

それに対して、\$DOUBLE 形式の数値の範囲は、ネイティブ形式で許可される 1.0E145 よりもはるかに大きい 1.0E308 になります。数値の範囲が重要となるアプリケーションでは、\$DOUBLE を使用する必要があります。

データを外部で共有するアプリケーションでは、暗黙の変換対象とならないよう、データを \$DOUBLE 形式で維持することも検討できます。他のほとんどのシステムでは、基礎となるハードウェア・アーキテクチャによって直接サポートされることから、バイナリ浮動小数点の表現に IEEE 標準を使用しています。したがって 10 進形式の値は、(例えば ODBC/JDBC、SQL、または言語バインディング・インタフェースを使用して) 情報交換前に変換する必要があります。

InterSystems IRIS の 10 進数に定義された範囲内に \$DOUBLE 値がある場合、10 進数に変換して再度 \$DOUBLE 値に戻すと、常に同じ数値が生成されます。\$DOUBLE 値の精度は 10 進値よりも低いため、逆の場合はこのようにはなりません。

その理由から、可能であれば、いずれか 1 つのデータ表現のみを使用して計算を実行することをお勧めします。データ表現間で変換を繰り返すと、精度が落ちる場合があります。多くのアプリケーションでは、すべての計算に InterSystems IRIS の 10 進形式を使用できます。\$DOUBLE 形式は、IEEE 形式を使用するシステムとデータを交換するアプリケーションのサポートを目的としています。

\$DOUBLE ではなく、InterSystems IRIS の 10 進形式をお勧めする理由は、以下のとおりです。

- ・ InterSystems IRIS の 10 進形式は精度が高く、\$DOUBLE の桁数が 16 未満であるのに対し、ほぼ 19 桁の精度を持っています。
- ・ InterSystems IRIS の 10 進形式では、正確に 10 進数の小数を表現できます。例えば、0.1 という値は InterSystems IRIS の 10 進形式では正確に 0.1 になりますが、バイナリの浮動小数点では同値となる値がないので、\$DOUBLE 形式では語義的な近似値で 0.1 を表現する必要があります。

科学的数値では、以下の点でインターシステムズの 10 進形式よりもインターシステムズの \$DOUBLE の方が有利です。

- ・ \$DOUBLE は、ほとんどの計算ハードウェアで使用されている IEEE 倍精度浮動小数点とまったく同じデータ表現を使用します。
- ・ \$DOUBLE の方が範囲が広く、\$DOUBLE による最大値は 1.7E308 であるのに対し、インターシステムズの 10 進形式による最大値は 9.2E145 です。

D.3 数値表現の変換

注意 インターシステムズでは、10 進形式と \$DOUBLE 形式間の変換は、アプリケーションで明示的に制御することを推奨しています。

値を文字列から数値に変換する場合や、記述された定数をプログラムのコンパイル時に処理する際、最初の 38 桁の有効桁数のみが仮数の値に影響します。それ以降のすべての桁はゼロとして処理されます。つまり、指数値の決定に使用され、仮数値には影響しません。

D.3.1 文字列

D.3.1.1 数値としての文字列

InterSystems IRIS では、式に文字列が使用されている場合、その文字列の値は、その最初の文字で開始される文字列内の最も長い数値リテラルになります。そのようなリテラルが存在しない場合、文字列の計算値は 0 になります。

D.3.1.2 添え字としての数値文字列

計算では、“04”と“4”の文字列間に違いはありません。しかし、このような文字列がローカル配列またはグローバル配列の添え字として使用される場合、InterSystems IRIS ではこれらの文字列が区別されます。

InterSystems IRIS では、先頭（存在する場合はマイナス記号の後）や 10 進小数の末尾に 0 を含む数値文字列は、添え字として使用する場合、文字列であるかのように処理されます。これらは、文字列として、数値を含んでいるため、計算に使用できます。しかし、ローカル変数またはグローバル変数の添え字としては、文字列として扱われ、文字列として照合されます。以下に、例をいくつか示します。

- ・ “4”と“04”
- ・ “10”と“10.0”
- ・ “.001”と“0.001”
- ・ “-.3”と“-0.3”
- ・ “1”と“+01”

左側は添え字として使用した場合、数値と見なされ、右側は文字列として扱われます。（無関係な先頭と末尾の 0 の部分を除く左側の形式は、“キャノンニック”形式とも呼ばれます。）

通常の照合では、以下の例のように、数値は文字列の前にソートされます。

ObjectScript

```
SET ^||TEST("2") = "standard"
SET ^||TEST("01") = "not standard"
SET NF = "Not Found"

WRITE ""2"", ": ", $GET(^||TEST("2"),NF), !
WRITE 2, ": ", $GET(^||TEST(2),NF), !
WRITE ""01"", ": ", $GET(^||TEST("01"),NF), !
WRITE 1, ": ", $GET(^||TEST(1),NF), !, !
SET SUBS=$ORDER(^||TEST(""))
WRITE "Subscript Order:", !
WHILE (SUBS != "") {
    WRITE SUBS, !
    SET SUBS=$ORDER(^||TEST(SUBS))
}
```

D.3.2 10 進数から \$DOUBLE への変換

\$DOUBLE 形式への変換は、\$DOUBLE 関数を使用して明示的に実行されます。また、この関数は、\$DOUBLE(<S>) 式を使用して、非数および無限大に対する IEEE 表現の明示的な構築を許可します。ここで <S> は、以下を意味します。

- ・ 文字列 “nan” は非数を生成します。
- ・ 文字列 “inf”、“+inf”、“-inf”、“infinity”、“+infinity”、または “-infinity” のいずれかは、無限大を生成します。

- ・ 数値リテラルおよび文字列リテラルは、それぞれ -0 および “-0” を生成します。

注釈 文字列 <S> のケースは、入力時に無視されます。出力時には、“NAN”、“INF”、および “-INF” のみが生成されます。

D.3.3 \$DOUBLE から 10 進数への変換

\$DOUBLE 形式の値は、\$DECIMAL 関数で 10 進数値に変換されます。この関数の呼び出しの結果、10 進数値への変換に適した文字列が生成されます。

注釈 この説明では、\$DECIMAL に指定される値は \$DOUBLE 値であることを前提としていますが、必ずしもそうである必要はありません。任意の数値を引数として指定でき、丸めにも同じルールが適用されます。

D.3.3.1 \$DECIMAL(x)

引数を 1 つ持つ形式のこの関数は、引数として指定された \$DOUBLE 値を 10 進数に変換します。\$DECIMAL によって数値の小数部は 19 桁に丸められます。\$DECIMAL では常に最も近い 10 進数値に丸められます。

D.3.3.2 \$DECIMAL(x, n)

引数を 2 つ持つ形式では、返される桁数を正確に制御できます。n が 38 より大きい場合、<ILLEGAL VALUE> エラーが生成されます。n が 0 より大きい場合、n 桁の有効桁数に丸められた x の値が返されます。

n が 0 の場合、値の決定には、以下のルールが使用されます。

1. x が無限大の場合、必要に応じて “INF” または “-INF” を返します。
2. x が非数の場合、“NAN” を返します。
3. x が正または負の 0 の場合、“0” を返します。
4. x を 20 桁以下の有効桁数で正確に表現できる場合、その有効桁数のキャノン形式の数値文字列を返します。
5. それ以外の場合は、10 進表現を 20 桁の有効桁数に切り捨てます。さらに
 - a. 20 番目の桁が “0” の場合、“1” に置き換えます。
 - b. 20 番目の桁が “5” の場合、“6” に置き換えます。

その後、結果の文字列を返します。

20 桁目が不正確に “0” または “5” になる場合を除き、20 桁目を 0 に切り捨てるこの丸めのルールには、以下の特性があります。

- ・ ある \$DOUBLE 値が、ある 10 進数値と異なる場合、これらの 2 つの値は常に、等しくない表現文字列を持つことになります。
- ・ <MAXNUMBER> エラーを生成せずに \$DOUBLE 値を 10 進数に変換できる場合、その結果は \$DOUBLE 値を文字列に変換してから、その文字列を 10 進数値に変換する場合と同じです。2 つの変換を行う場合、“double round” エラーは発生しません。

D.3.4 10 進数から文字列への変換

10 進数値は、例えば連結演算子のオペランドの 1 つとして使用した場合、既定で文字列に変換できます。変換を詳細に制御する必要がある場合には、\$FNUMBER 関数を使用します。

D.4 数値を含む演算

D.4.1 算術演算子

D.4.1.1 同種の表現

10 進数値のみを含む式では、常に 10 進数の結果が生成されます。同様に、\$DOUBLE 値のみを含む式では、常に \$DOUBLE の結果が生成されます。さらに、

- ・ 10 進数値を含む計算結果がオーバーフローする場合、<MAXNUMBER> エラーとなります。リテラルの場合のように、\$DOUBLE への自動変換は行われません。
- ・ 10 進数式がアンダーフローする場合、式の結果として 0 が生成されます。
- ・ 既定では、オーバーフロー、0 による除算、および無効な演算の IEEE エラーは、無限大や非数の結果を生成するのではなく、それぞれ <MAXNUMBER>、<DIVIDE>、および <ILLEGAL VALUE> エラーとして示されます。この動作は、個別プロセスの場合は %SYSTEM.Process クラスの IEEEError() メソッド、システム全般の場合は Config.Miscellaneous クラスの IEEEError() メソッドで変更できます。
- ・ $0 ** 0$ (10 進数) という式では 10 進数値の 0 が生成されますが、 $\$DOUBLE(0) ** \$DOUBLE(0)$ という式では \$DOUBLE 値の 1 が生成されます。前者は InterSystems IRIS では常に当てはまっています。後者は IEEE 標準の要件です。

D.4.1.2 異種の表現

10 進数表現と \$DOUBLE 表現の両方を含む式では、常に \$DOUBLE 値が生成されます。値の変換は、使用時に実行されます。したがって、次のような式の場合、

```
1 + 2 * $DOUBLE(4.0)
```

InterSystems IRIS は最初に、10 進数値として 1 と 2 を加算します。その後、結果の 3 を \$DOUBLE 形式に変換し、乗算を実行します。結果は \$DOUBLE(12) です。

D.4.1.3 丸め

必要に応じて、数値は最も近い表現可能な値に丸められます。丸められる値が、2 つの使用可能な値と同等に近い場合、以下が実行されます。

- ・ \$DOUBLE 値は、IEEE 標準で定義されているとおりに、偶数に丸められます。
- ・ 10 進数値は、0 から離れた値、つまり (絶対値で) 大きい方の値に丸められます。

D.4.2 比較演算子

D.4.2.1 同種の表現

\$DOUBLE(+0) と \$DOUBLE(-0) の比較では、これらの値は同等に扱われます。これは IEEE 標準に準拠しています。\$DOUBLE(+0) または \$DOUBLE(-0) が文字列に変換される場合、両方とも結果が “0” になるので、これは InterSystems IRIS の 10 進数の場合と同じです。

\$DOUBLE(“nan”) と他の数値 (\$DOUBLE(“nan”) を含む) の比較では、これらの値は、より大きくない、等しくない、より小さくない、になります。これは IEEE 標準に準拠しています。これは、文字列に変換してからそれらの文字列が等しいかどうかを確認することにより等値比較を行うという、通常の ObjectScript のルールから逸脱しています。

注釈 式 “nan” は、\$DOUBLE(“nan”) と同等です。これは、比較が文字列の比較として行われるためです。

注釈 ただし、\$LISTSAME では、\$DOUBLE(“nan”) を含むリスト要素を、\$DOUBLE(“nan”) を含むリスト要素と同一と見なします。この場合にのみ、\$LISTSAME は、等しくない値を等しいと見なします。

D.4.2.2 異種の表現

10 進数値と \$DOUBLE 値の比較は、完全に正確です。比較は、いずれの値も丸められることなく実行されます。有限値のみが含まれる場合、これらの比較の答えは、両方の値を文字列に変換し、これらの文字列を既定の照合ルールに基づいて比較した場合に得られる結果と同じになります。

演算子 <, <=, >, および => を含む比較では常に、ブーリアン値、0 または 1 を 10 進数値として生成します。いずれかのオペランドが文字列である場合、そのオペランドは、比較を実行する前に 10 進数値に変換されます。他の数値オペランドは変換されません。前述のとおり、異なる数値型の比較は、完全に正確に実行され、変換は行われません。

文字列の比較演算子 (=, '=:], '], [, '[,], ']] など) では、すべての数値オペランドは、比較を行う前にまず文字列に変換されます。

D.4.2.3 以下、以上

InterSystems IRIS では、演算子 “<=” および “>=” はそれぞれ、演算子 “>” および “<” と同様に扱われます。

注意 オペランドのいずれかまたは両方が非数である可能性がある比較で、演算子 “<=” または “>=” が使用される場合、IEEE 標準の要求とは異なる結果になります。

A と B のいずれか (または両方) が非数の場合、“A >= B” という式は、以下のように解釈されます。

1. 式は “A > B” に変換されます。
2. さらに “!(A < B)” に変換されます。
3. 前述のように、非数を含む比較では、(a) 等しくない、(b) より大きくない、(c) より小さくない、という結果が得られます。したがって、括弧の中の式の結果は、False の値になります。
4. その値を反転させると、True の値が得られます。

注釈 “A >= B” という式は、これを “((A > B) | (A = B))” と表現すれば、書き換えて IEEE で要求される結果を得ることができます。

D.4.3 ブーリアン演算

ブーリアン演算 (and, or, not, nor, nand など) では、すべての文字列オペランドが 10 進数に変換されます。数値オペランド (10 進数または \$DOUBLE) は変更されません。

数値 0 は FALSE として扱われ、その他のすべての数値 (\$DOUBLE(“nan”) を含む) および \$DOUBLE(“inf”) は TRUE として扱われます。結果は 0 または 1 (10 進数) です。

注釈 \$DOUBLE(-0) も False です。

D.5 値の正確な表現

各数値形式には、保持できる有効桁数の限度があります。普通は、この限度よりも桁数が多ければ、自動的に精度が落ちると考えがちです。しかし、一般的にはこれは正しくありません。

この影響は、InterSystems IRIS の 10 進数値に最もよく見られます。単純に、値 9,223,372,036,854,775,807 が正確に表現できる最大の数値であると考えがちですが、その考えでは指数が無視されています。9,223,372,036,854,775,807,000,000 は、百万倍大きいですがこれも正確に表現できることは明らかです。値が許可された範囲内にある、指数が 0 の他のすべての数値について、同じことが当てはまります。しかし、これは、10 進数の全範囲内で取り得るすべての数値ではありません。

この表現では、10 の累乗の代わりに 2 の累乗を数値に掛けること以外、状況は似ています。以下の特徴がある非常に単純なモデルを考えれば、この状況を図で示すことができます。

- ・ 常に正の数を扱うので、符号ビットは必要がない。
- ・ 3 ビットの非正規化された仮数 (値の範囲は 0 ~ 7) を使用する。
- ・ バイアスがない符号付の 3 ビットの指数 (値の範囲は -4 ~ +3) を使用する。

以下の表は、表現可能な値を示します。

指数	-4	-3	-2	-1	0	1	2	3
仮数								
0	0	0	0	0	0	0	0	0
1	0.0625	0.125	0.25	0.5	1	2	4	8
2	0.125	0.25	0.5	1	2	4	8	16
3	0.1875	0.375	0.75	1.5	3	6	12	24
4	0.25	0.5	1	2	4	8	16	32
5	0.3125	0.625	1.25	2.5	5	10	20	40
6	0.375	0.75	1.5	3	6	12	24	48
7	0.4375	0.875	1.75	3.5	7	14	28	56

以下の点に注意してください。

- ・ このモデルでは、仮数には 3 ビットしか使用しないので、1 桁の 10 進数には十分な精度がありません。9 という数値がありません。これを表現するには、仮数として 1001 が必要となり、これは明らかに長すぎるからです。
- ・ これらのバイナリ数値をすべて完全に正確な 10 進数文字列と正確に対照させるには、表現で有効な 4 桁の 10 進数を使用する必要があります。
- ・ 0 ~ 56 の範囲の数値には表現できないものがあります。一例として 2.75 という値が挙げられます。段階的に仮数を整数に変換すると、結果として 5.5、次に 11 という値が得られる一方で、これに対応して、指数が -1、次に -2 となります。しかし 11 は 3 ビットでは表現できない値なので、上記の表内にはありません。

以下の図では、この形式で表現できる値が、数直線上においてどのように分布しているかを示しています。図示するために、刻み幅を、最小の表現可能な値である 0.0625 (つまり 1/16) としています。数直線は、各直線が 4 単位分の長さ (つまり 64 刻み) となるように、折り返されています。X は、この形式で表現できる数値を示しています。配置可能な 56*16=896 個の位置のうち、正確に表現できるのは 64 個 (平均では 14 個中 1 個) のみです。

	+0	+1	+2	+3
0	X	X	X	X
4	X	X	X	X
8	X	X	X	X
12	X	X	X	X
16	X	X	X	X
20	X	X	X	X
24	X	X	X	X
28	X	X	X	X
32	X	X	X	X
36	X	X	X	X
40	X	X	X	X
44	X	X	X	X
48	X	X	X	X
52	X	X	X	X
56	X	X	X	X

同様な状況が、InterSystems IRIS の 10 進数値にも当てはまります。正確に表現できる (指数が 0 の) 最大の整数は、9,223,372,036,854,775,807 です。説明の都合上、この値を MAX と呼びます。この結果、MAX*10 および (MAX-3)*100 も正確に表現できることになります。これらの指数は 0 ではありませんが、仮数は正確です。ただし、MAX+1 などのように正確に表現できない値もあります。

\$DOUBLE 値でも、指数が InterSystems IRIS の場合のように 10 ではなく 2 の累乗である場合を除き、この現象が発生します。

D.6 関連項目

詳細は、以下のソースを参照してください。

- ・ [IEEE-754-1985](#) 標準。この標準の正式名は、“IEEE Standard for Binary Floating-Point Arithmetic”です。米国では、この仕様は ANSI/IEEE Std 754-1985 と表記されます。
これは国際標準でもあります。国際的には、IEC 60559:1989 “Binary floating-point arithmetic for microprocessor systems” と呼ばれています。
- ・ Computing Surveys の David Goldberg 氏著による “[What Every Computer Scientist Should Know About Floating-Point Arithmetic](#)” (1991 年 3 月発行)。

