

序文

Cache Basic チュートリアルへようこそ。

このチュートリアルを終了すると、Cache 特有の拡張を含め、インターシステムズが実装する一般的な言語である Basic を学習できます。

Cache Basic を使用して、すばやく高度なデータベース・アプリケーションを構築できます。その方法については以下を参照してください。

- 複数值文字列の処理
- スパース配列との連動
- 複雑で現実的なリレーションシップを持つ、大規模なデータベースの生成と管理
- データベース・オブジェクトの簡単な操作

第1章

はじめに

このページは、VBScript に類似した Cache Basic の導入部分です。ここでは、Visual Basic あるいは VBScript を既に理解していることを前提にしているため、一般的なコマンドや関数に関する説明はありません。

第 1 章では、Cache Basic を強力にしている以下の機能について学習します。

- 複数值文字列が持つ多くの機能
- Cache スタジオ・デバッガの使用法
- 配列、ツリー構造、グローバル変数の存在のテスト法
- データベースとインデックスの簡単な作成方法

すべてのCacheコマンド、関数、定数を参照するには[Cacheドキュメント: \[Cache開発リファレンス\]-\[Cache Basic リファレンス\]](#) を参照してください。

チュートリアル演習

このチュートリアルでは、Cacheスタジオを使用していくつかのプログラムを作成し、Cache Basic のすべての特徴を体験することができます。また、コマンドや関数の使用法の例もあります。

Cacheスタジオを使用して小規模なプログラムを作成し、これらの例を試してください。

Cache Basic を学ぶ一番よい方法は、チュートリアルすべてを含む練習問題を実践することです。(理解力をテストするために)独自に挑戦することもできますし、段階的な指示に従って進めることもできます。

Caché Basic ルーチン

ここでは、簡単ですが重要な Caché Basic ルーチンについて学習します。RightTriangle ルーチンのコードは、[Appendix C: Right Triangle ルーチン] をご参照下さい。このルーチンでは、直角三角形の両辺の長さを元に面積と斜辺を計算します。では、見てみましょう。

```
SAMPLES>do Run^BASRightTriangle()
```

```
Compute the area and hypotenuse of a right triangle  
given the lengths of its two sides.
```

```
First, choose a unit of measurement.
```

```
(i)nches, (f)eet, (m)iles, (c)entimeters, m(e)ters, (k)ilometers: i
```

```
Length of side 1: 3 Accepted.
```

```
Length of side 2: 4 Accepted.
```

```
The area of this triangle is 6 square inches.
```


```
The hypotenuse is 5 inches.
```

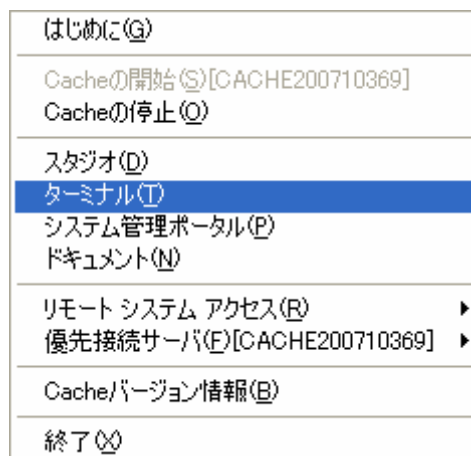
```
SAMPLES>
```

コードからわかるように、ルーチンには、パブリックとプライベートの両方のサブルーチンと関数があり、また、一般的なコマンドと関数 **PrintIn**、**Input**、**Return**、**Rxit Sub/Do**、**Select Case**、**Do/Loop**、**If/Else**、**Left()**、**Chr()**、**Round()**、**Sqr()** が含まれています。

次に、Caché Basic を使用してみましょう。

ターミナル

Basic を使用してチュートリアルを開始するには、Cache ターミナルを起動する必要があります。タスク・バーの Cache キューブ・アイコンを右クリックし、メニューから **【ターミナル】** を選択します。ターミナル・ウィンドウが表示され、そのプロンプトから現在は USER ネームスペースにいたことがわかります。ネームスペースは論理的なディレクトリで、ルーチン(プログラム)とグローバル(データ)を含みます。



Cache ターミナルで Basic コマンドを実行すると、瞬時に結果が表示されます。この章では、大半のページにターミナル・セッションを含んでいます。最初に説明文を記述し、その下部にコード例(通常は実行可能)を示しています。また、Cache スタジオを使用してルーチンを記述します。

ターミナルの使用法

Caché ターミナルは、Basic 環境とインタラクティブではありません。 Caché ObjectScript とは事実上インタラクティブです。しかし、ObjectScript は Basic を、あるいは Basic は ObjectScript を呼び出すことができます。 ObjectScript の **Do** と **Write** コマンドを使用して、ターミナルから Basic ルーチンを実行できます。

Caché ターミナルを開始すると、*USER* ネームスペースに移ります。do ^%CD を使用し、*SAMPLES* ネームスペースに移動します。ここでは、提供された例を参考にし、自身の演習を実行できます。

```
USER>do ^%CD

Namespace: SAMPLES
You're in namespace SAMPLES
Default directory is c:\¥cachesys¥mgr¥samples¥
SAMPLES>

SAMPLES>
```

ObjectScript コマンド

Caché ターミナルを使用して Basic のサブルーチンをテストするには、構文 `do sub^routine (args)` を使用します。

Caché ターミナルを使用して Basic の関数をテストするには、構文 `write $$func^routine (args)` を使用します。

ObjectScript コマンド **Do** と **Write !** は、Basic コマンドの **Call** と **Println** と類似しています。**Do** コマンドは、前ページの例で既に **BASRightTriangle** ルーチンを実行するために使用しました。以下の例は、**BASRightTriangle** ルーチン内で **IsNegative()** 関数を呼び出す **Write** コマンドです。

```
write !, $$IsNegative^BASRightTriangle(4)
write !, $$IsNegative^BASRightTriangle(-4)
```

変数

Cache Basic 変数は変動します。オプションとして、**Dim** を使用して宣言します。割り当てられない変数の値は空文字列 ("") です。

```
x = 4 + 2
println "result = ", x
println "the empty string = ", y
```

Option Explicit を指定し変数を宣言すると便利です。コンパイル時、Cache スタジオは未定義のまま使用されている変数にフラグを立てます。

サブルーチン外あるいは関数外で宣言されたモジュール・レベルの変数は、ルーチンに対してグローバルです。サブルーチン内あるいは関数内で宣言された場合はローカルです。

ディスクへのデータの格納

通常の変数を使用するのと同様に、簡単にデータをディスクに永続的に格納できます。このためには、変数名の先頭に曲折アクセント記号(^)を付けます。以下の例では、通常の変数 `x` とは異なる変数 `^x` で文字列を保存します。詳細は後で学習します。

コードを実行するたびに、更新する前の `^x` の現在値を表示します。

```
x = 4
println "x = ", x
println "old value of ^x = ", ^x
^x = "random number: " & int((rnd()*10))
println "new value of ^x = ", ^x
```

複数値文字列

Cache 独特の機能は、複数値文字列と簡単に連動できることです。変数は 32K までのデータを含むため、1 つの変数を使用して複数の値の文字列(部分文字列)を保持できます。文字列はレコードとして、部分文字列はそのフィールドとして機能します。区切り文字列とリストの 2 種類の方法があり、区切り文字列は、値を区切るために“^”のような指定の区切り文字を使用します。リストも同様ですが、区切り記号は内部的に管理されるため、プログラマは関与しません。

- 連結を介して区切り文字列を構築します。 **\$Piece** は、区切り記号を使用して文字列を区分けし、特定の文字列を返します。 **Len()** の 2 つの引数形式は、区切り記号を使用して、区分けされた文字列の数を返します。 **InStr()** を使用して、文字列の一部を検索します。
- **ListBuild()** を使用してリストを構築します。 **List()** を使用してリストに項目を返します。 **ListLength()** を使用してリスト内の項目数を返し、 **ListFind()** は項目を検索します。

区切り文字列

Piece()には他にも特徴があります。単一の文字ではなく、各種部分文字列を返します。異なる区切り文字を使用して、同じ文字列から部分文字列を返すこともできます。**Piece()**は戻り値が空の場合、空文字列を返します(最初の区切り文字の前に値がない、あるいは区切り文字が2つ連続していない場合)。

```
println "using string: slice of pizza"
println "3rd piece = ", piece("slice of pizza", " ", 3)
println "1st and 2nd pieces = ", piece("slice of pizza", " ", 1, 2)
println "2nd piece by i = ", piece("slice of pizza", "i", 2)
println "2nd (empty) piece by z = ", piece("slice of pizza", "z", 2)
println "count of pieces = ", len("slice of pizza", " ")
address = "One Memorial Drive^Cambridge^MA^02142"
city = piece(address, "^", 2)
println "city = ", city
println "MA starts at position #", InStr(address, "^MA^")+1 ' one past ^
```

リスト

区切り文字を使用したくない場合に、リストは便利です。部分文字列は通常、区切り文字列を含み、部分文字列のリスト内の位置を返します。リストはこれを防ぎます。

Piece()と同様に、**List()**は他にも特徴があります。単一の項目ではなく、項目リストを返します。空の部分文字列が存在する場合、**List()**は<NULL VALUE>エラーを生成し、**ListGet()**は空文字列を返します。

```
addr = "One Memorial Drive" : city = "Cambridge" : st = "MA" : zip = "02142"
mail = listbuild(addr, city, st, zip)
println "city = ", list(mail, 2)
println "5th (empty) item = ", listget(mail, 5)
cityst = list(mail, 2, 3)
println "state = ", list(cityst, 2)
println "count of items = ", listlength(mail)
println "MA is item #", listfind(mail, "MA")
```

通常の文字列では、List 関数は<LIST>エラーが検出されるため、使用できません。同様に、リストでは文字列関数を使用しません。**\$ListBuild** で構築される文字列は **Println** コマンドで表示できますが、制御文字を含むため、お勧めできません。したがって、専用の List 関数を使用してリストにアクセスするようにします。

一般的なエラー

Cacheは誤った入力に対し、エラー・メッセージを返します。エラー・メッセージの詳細情報は、[Cacheドキュメント:\[Cache開発リファレンス\]-\[Cache エラー・リファレンス\]-\[Cacheシステム・エラー・メッセージ\]](#) に記述されています。以下は、一般的な 3 つのエラーです。

- <NOROUTINE>は存在しないルーチンを実行しようとした場合に返されます。
- <NOLINE>は存在しないサブルーチンを呼び出そうとした場合に返されます。
- <PARAMETER>は必要なパラメータを渡さなかった場合に返されます。

```
SAMPLES>do Run^BASRightTri ()

D Run^BASRightTri ()
^
<NOROUTINE>
SAMPLES>do RAN^BASRightTriangle ()

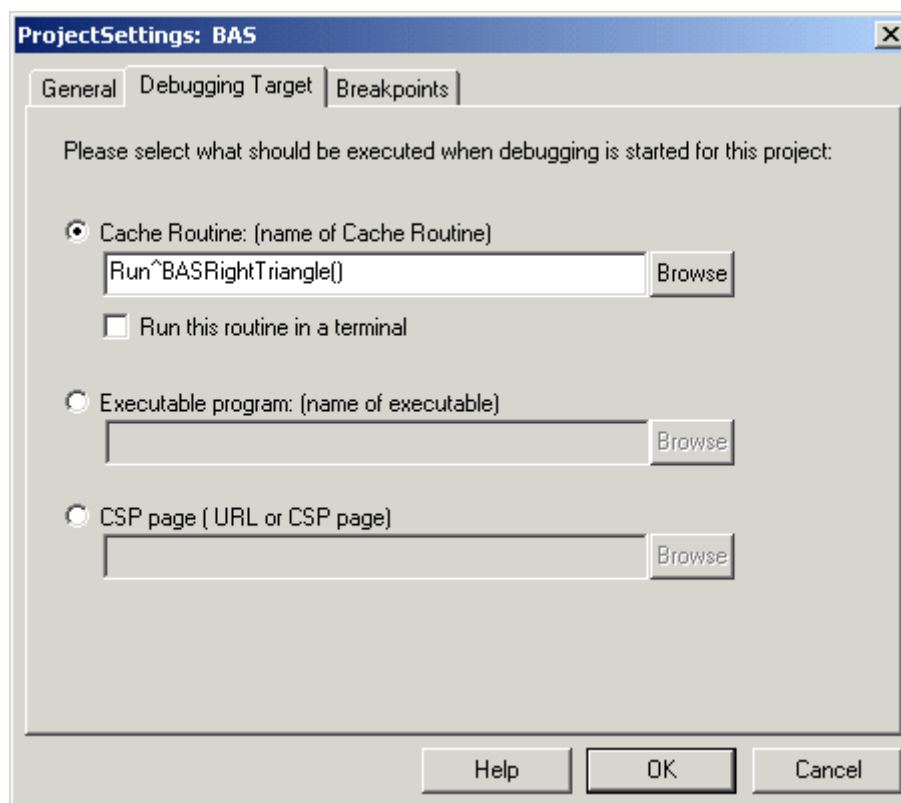
D RAN^BASRightTriangle ()
^
<NOLINE>
SAMPLES>do Run^BASRightTriangle

Sub Run ()
^
<PARAMETER>+4^BASRightTriangle
SAMPLES 2d0>Quit

SAMPLES>
```

3 番目のエラーは、**BASRightTriangle** ルーチンの実行中に発生します。これにより、ルーチンの実行は終了します。エラー・メッセージは、Cache ターミナル・ウィンドウでエラーを含む行(オフセットを追加したラベル名)に表示されます。また、**SAMPLES>**プロンプトは **SAMPLES 2d0>**に変更されます。ObjectScript の **Quit** コマンドを使用して、通常の状態にプロンプトを戻します。

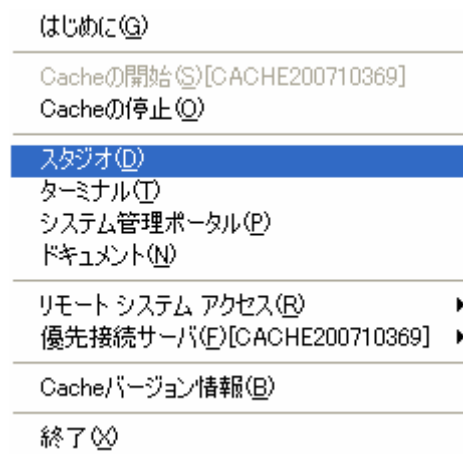
Caché スタジオは、Basic にインタラクティブなデバッガを提供します。プロジェクトの設定内で、**[プロジェクト]**→**[設定]**をクリックし、常時デバッグしたいルーチンを **[デバッグ対象]**として指定します。



F9 を使用すると、ルーチンで 1 つ以上のブレークポイントを切り替えることができます。その後、**F5** を押し、デバッガを使用してルーチンを実行します。 Caché スタジオは、Caché ターミナル・プロセスを生成し、アタッチします。ルーチンは、そのプロセス内で実行します。すべての出力を表示し、すべての入力が可能です。スタジオで、**F10**(ステップオーバー)と**F11**(ステップイン)を使用し、コードを行き来できます。**[ウォッチ]**ウィンドウを使用すると、変数を指定し、コードの実行により変更した値を表示できます。

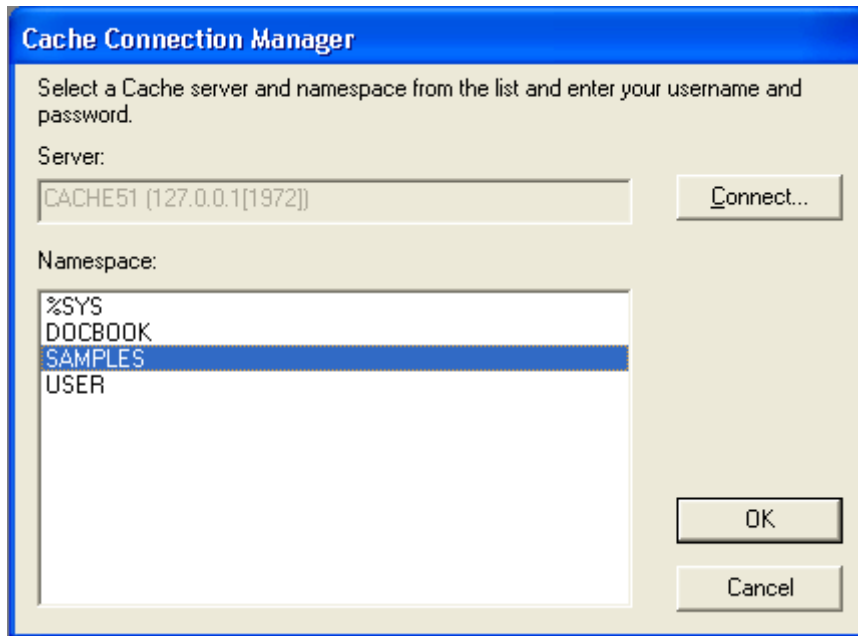
ルーチンの記述

Cache Basic の記述を実際に始めるために、備え付けの簡単なデータ入力ルーチンから開始します。演習過程で、その内容を変更していきます。この演習は、Cache スタジオ開発環境の導入としても役立ちます。まず、タスク・バーにある Cache のキューブ・アイコンを右クリックし、メニューから **【スタジオ】** を選択してください。



Cache スタジオ

Cache スタジオを使用するときは、最初にネームスペースを選択します。Cache ターミナルで行ったように、[SAMPLES]を選択します。Cache スタジオを以前使用したことがある場合は、最後に使用したネームスペースに接続します。SAMPLES ネームスペースに変更するには、**[ファイル] → [ネームスペース変更]**をクリックします。



接続すると、Cache スタジオ・インタフェースや Project1 と呼ばれる既定(空)プロジェクトが表示されます。

このチュートリアルで使用する例はすべて、BAS プロジェクトや SAMPLES で参照できます。これをロードするには、**[ファイル] → [プロジェクトを開く]**をクリックし、[BAS]を選択します。プロジェクトを開くと、**[ルーチン]**フォルダに含まれるソース・コードを表示します。

[ファイル] → [プロジェクトの新規作成]をクリックして、新しいプロジェクトを作成します(Project1に戻ります)。**[ファイル] → [最近使用したプロジェクト]**を使用すると、このプロジェクトとBASプロジェクトの間を簡単に行き来することができます。保存する際、MyWork のような名前をプロジェクトに付けることができます。

演習 1

最初の演習として、備え付けのデータ・エントリ・ルーチンをコピーします。その後、そのルーチンに簡単なエラー・チェックとメッセージを追加します。以下の仕様に従います。

- 3つのプロンプト、Name、Phone (US Style)、Date of Birth が表示されます。
- 以下の形式でデータを入力します。

```
Name:    John Smith
Phone:   555-1111
DOB:     1/23/45
```

- ユーザが Name プロンプトにデータを入力するまでプロンプトはループします。

以下の手順を実行します。

- Caché ターミナル・セッションを開始し、SAMPLES ネームスペースに移動します。その後、**Do** コマンドを使用しルーチンを実行します(`do main^BASdatent1()`)。
- Caché スタジオを使用して、**[ファイル]**→**[開く]**をクリックします。**[ファイルの種類]**フィールドで“[Basic ファイル]”を選択します。BASdatent1.BAS をダブルクリックして開き、プロジェクトにロードします。
- **[ファイル]**→**[プロジェクトを名前をつけて保存]**をクリックし、[myBASdatent.BAS]という名前でそのルーチンを保存します。
- **[Workspace]**ウィンドウの**[Project]**タブで**[ルーチン]**フォルダをダブルクリックし、BASdatent1.BAS を右クリックして削除します(プロジェクトからは削除しません)。
- **Piece()**を使用したパブリックの関数を 2 つ記述し、Name と Phone を検証します。無効なデータには、エラー・メッセージを返します。コードは、“Last,First”で名前を検証する必要があります。
- パブリックな **formatName()** 関数を記述すると、**UCase()**と **LCase()**関数を使用して、ユーザが入力した名前に関係なく姓と名を適切な名前の形式に変換します。名前の検証後にこの関数を呼び出します。
- Phone プロンプトに既定のエリア・コードを追加し、またエリア・コードを持たない電話番号には、この既定コードを割り当てる関数を行に追加します。
- 誕生日を有効にするため、**DateConvert()**と **DateDiff()**を使用するパブリック関数を記述し、日付を内部形式に変換します。無効な日付や(現在より)未来の日付の場合、エラー・メッセージを生成します。
- プロンプトから新しい関数を呼び出します。
- ユーザが実際入力した形式にかかわらず、日付を標準形式で表示します(**DateConvert()**を再度使用)。

- [ファイル]→[プロジェクトを保存]をクリックし、[ビルド]→[コンパイル]でルーチンをコンパイルします。
- Caché ターミナルで **Do** を使用し、ルーチンを実行します。

詳細は、[Appendix D: 演習 1 : データ入力ルーチンの変更] を参照してください。

Cache 配列

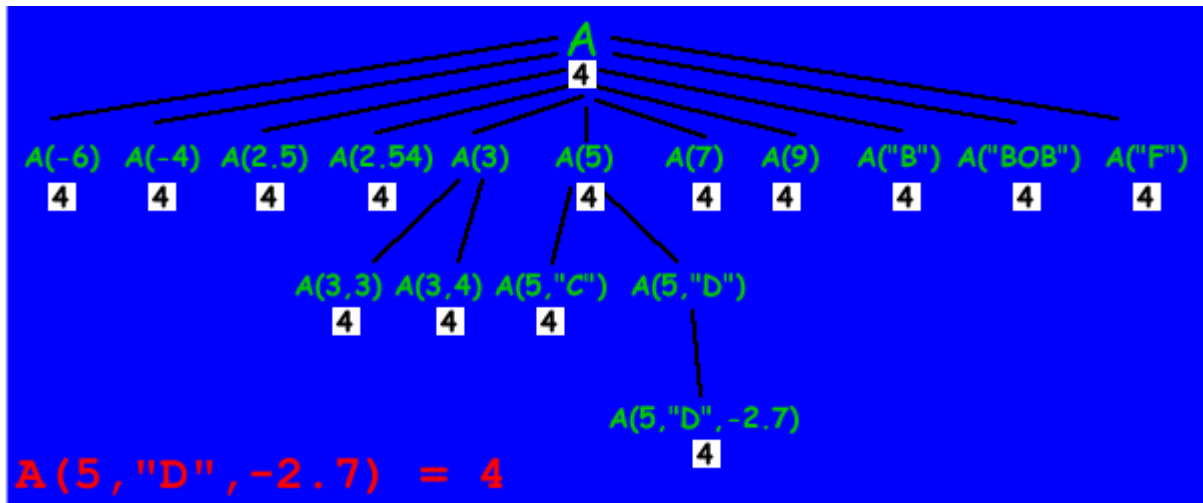
他言語と同様、Cache Basic は添え字を使用した多次元配列を備えています。しかし Cache Basic の配列は他言語と異なり、より強力な機能を備えています。添え字なしの変数と同様に、サイズや次元にかかわらず配列を宣言する必要はありません。配列はスパース構造(一部の要素がない)であるため、要素にデータが定義されていない場合、スペースは割り当てません。例えば、配列を使用して、チェッカーと呼ばれるゲームでのプレイヤーの区画を追跡するとします。チェッカー・ボードには 8x8 のマスがあります。他言語では、チェッカーが 24 のポジションしか占めていない場合でも、配列 Board(8,8)には 64 のメモリ位置が必要です。

```
a(3, 7) = 4 + 2  
println a(3, 7)
```

他言語と同様、リストやテーブル形式で配列を配置します。ここでは単純な 3x3 の配列 A を表示しています。Cache Basic の持つ強力な配列機能を理解するには、配列のツリー構造を学習する必要があります。次ページでは、配列構造を動画で示しています。また、他の配列機能も学習します。

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

ツリー構造(1)



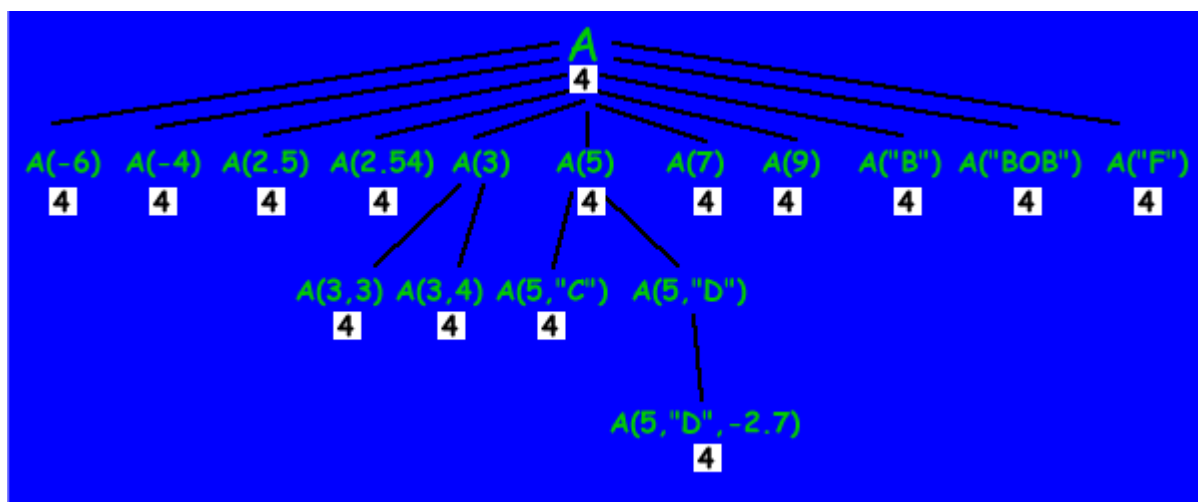
- ①set A = 4
- ②set A(5) = 4 ③set A(9) = 4 ④set A(7) = 4
- ⑤set A(-4) = 4 ⑥set A(-6) = 4
- ⑦set A(2.5) = 4 ⑧set A(2.54) = 4
- ⑨set A("B") = 4 ⑩set A("F") = 4 ⑪set A("BOB") = 4
- ⑫set A(3,3) = 4 ⑬set A(3,4) = 4 ⑭set A(5,"C") = 4
- ⑮set A(5,"D",-2.7) = 4

上記図では、最初に（添え字なしの） $A=4$ を設定しています<①>。ObjectScript は A にメモリ位置を割り当て、4 を格納します。この例では、すべてのロケーションに対して 4 を設定していますが、これは、任意です。重要なのは 4 という数値ではなく、これらの文によって構築されたツリーです。

次の 3 文は、複数の配列位置(5、9、7)に 4 を格納します<②~④>。 A は、スカラと配列の両方に使用できます。新規のレベルや新次元のツリーの配列は、文の順番に関係なく添え字値を元に並べ替えられます。この時点で、他言語では少なくとも 9 つのメモリ位置が必要ですが、 A では 4 つしか使用しません。次の 2 つの文では、負の整数添え字を使用します<⑤、⑥>。他言語では機能しませんが、ツリー構造の場合使用できます。また、実数や小数の添え字も Set 文で設定できます<⑦、⑧>。これもツリー構造でのみ使用可能な機能です。

次に、非数値の添え字を使用した 3 つの配列を設定します<⑨~⑪>。非数値の配列は数値配列の右側に表示されていますが、非数値はアルファベット順に並べ替えられます。ここまでのところ、A 配列は 1 次元です。次の 3 つの文で 2 次元配列を生成、並べ替え<⑫~⑭>、最後に 3 次元配列を生成します<⑮>。

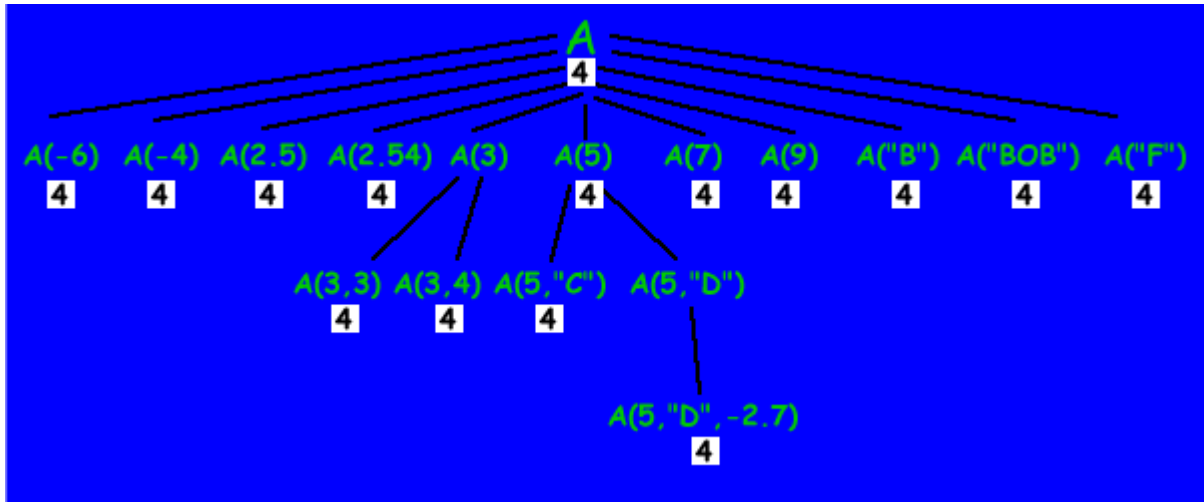
ツリー構造(2)



ここまで学習してきたように、正と負の整数、実数、非数値文字列を配列添え字として使用できます。空文字のみ使用できません。ここでは、ツリー構造の詳細について学習します。

配列に関連する用語を解説します。全体構造はツリーと呼ばれ、家系図のように上から下に拡張します。ルート A が頂点です。ルート、および他の添え字が付いた A をノードと呼びます。下にノードを持たないノードは、葉と呼ばれます。下にノードを持つノードは、親や先祖と呼ばれます。親を持つノードは、子や子孫と呼ばれます。また、同じ親を持つ子は兄弟と呼ばれます。すべての兄弟がツリーに追加されると、数値順あるいはアルファベット順で自動的に並べ替えられます。ツリー構造ではどのような深さの多次元配列も作成でき、現実的な問題を解決するための階層化が簡単にできます。他の方法で多次元配列を図示するのは困難です。

変数値とノードの存在、および Exists() 関数

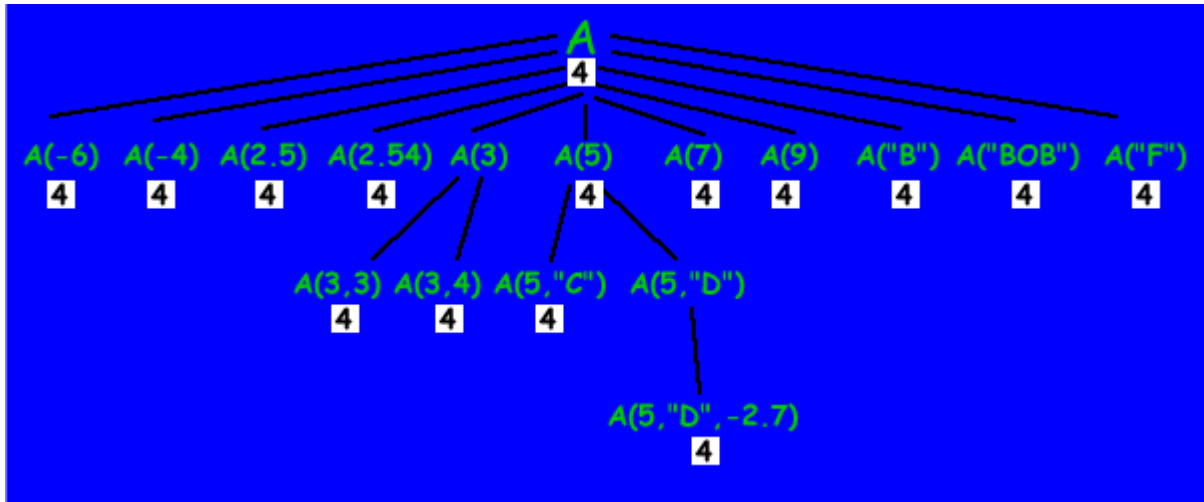


ノードの値はいつでも変更できます。しかし、値が割り当てられていないノードもあります。A(3)とA(5,"D")の2つには子があります。割り当ては下位に子を生じます。Print は、値のない変数やノードに空文字列を出力します。A(3)とA(5,"D")には値を割り当てないか、あるいは後で割り当てます。また、Basic 配列には別の機能も備わっています。つまり、メモリ位置は値を割り当てられていなくても存在します。

Exists()関数によって、ツリーのノードを検査しステータスを表示できます。Exists()がゼロ以外の結果を返した場合、ノードが存在していることを示します。コードを明確にするために、定数を持つExists()値を比較してください(ブラウザの [戻る]ボタンを使用してここに戻ることができます)。

```
A = 4 : A(3, 3) = 4
println exists(A), " means: value and descendants, equals ", _
    vbHasArray + vbHasValue
println exists(A(3)), " means: descendants only, equals ", _
    vbHasArray
println exists(A(3, 3)), " means: value only, equals ", _
    vbHasValue
println exists(A(3, 5)), " means: doesn't exist"
```

コマンド Erase、EraseValue、EraseArray



Erase、**EraseValue**、**EraseArray** コマンドは、配列内のパートやセクションを消去します。
Erase は、ノードとその子を消去します。特に、**Erase A** はその構造全体を消去します。
EraseValue は、ノードの値のみを消去します。**EraseArray** は、子ノードのみを消去します。

```
A = 4 : A(3) = 4 : A(3,3) = 4
print "A(3) before erasevalue: ", exists(A(3))
erasevalue A(3) : println " after: ", exists(A(3))
print "A(3) before erasearray: ", exists(A(3))
erasearray A(3) : println " after: ", exists(A(3))
print "A before erase: ", exists(A)
erase A : println " after: ", exists(A)
```

Caché のグローバルは、他言語のグローバルと意味が異なります。他言語のグローバル変数は、モジュール内にある複数のサブルーチンと関数にグローバルです。Caché のグローバル変数は、ディスクに保存されるため、すべてのプロセスに対してグローバルです。以下はその概要です。

用語	意味	アクセス範囲	例
グローバル	ディスク上の永続配列	ディスクにアクセスできるすべての Basic 処理と、すべてのルーチンはこの変数にアクセスできます。	[^] A
モジュール・レベル (グローバル) 変数	Basic ルーチン内、サブルーチン外、関数で定義される変数	モジュール内のサブルーチンと関数はこの変数にアクセスできます。	A
プライベート (ローカル) 変数	サブルーチンあるいは関数で定義される変数	変数を定義するサブルーチンあるいは関数のみがこの変数にアクセスできます。	A

Basic のグローバルを理解するのは非常に簡単です。これまで学習してきた配列はすべてグローバルです。配列は宣言する必要がなく、ディスクにデータを格納する場合、(グローバル名に“[^]”を使用して)割り当て文を使用するだけで済みます。前述のアニメーションで学習したように、ディスクにツリー構造を構築します。また、ディスク容量は値が割り当てられた場合にのみ必要です。3つの **Erase***関数は、グローバルのパートやセクションを消去します。その際は、グローバル全体(何百万のレコードを持つこともあります)を瞬時に消去します。通常の変数で使用する関数やコマンドのほとんどは、グローバルと同様に機能します。唯一、曲折アクセント記号“[^]”がグローバル名の先頭に表示される点が異なります。

Increment()関数

これまで学習してきた配列の構造や機能はユニークであるため、使用法に関して説明が必要でしょう。これらを整理するため、例として **myBASdatent** ルーチンにストレージ・プロシージャを記述します。ここでは簡単な **Increment()** 関数を学習します。\$Increment 関数は変数を取得し、値をインクリメント(変更)して返します。インクリメントは正でも負の数でも可能です。

Note:

Increment()関数は、その引数値を変更します。

```
println "current value of a = ", a
println "incremented to ", increment(a)
println "incremented to ", increment(a)
println "incremented to ", increment(a, 3)
println "decremented to ", increment(a, -2)
println "a = ", a
```

ストレージ・サブルーチン(1)

myBASdatent のストレージ・プロシージャについて学習します。これまで、プロンプトに応答しデータを確認しました。ここでは、データを保存するオプションが必要です。次のスライドで示しているように、グローバル ^PersonD にデータを格納します。

一人一人に新しい一意の ID 番号を生成して、^PersonD の添え字として使用します。

myBASdatent の初回実行時は ^PersonD が存在しないため、**Increment()** を使用して最初の ID を必ず 1 にする必要があります。これにより、“通常”の配列と同様の概念構造を持つ配列、つまり人のリストに ^PersonD が配置されます(ただし、実際はまだツリー構造です)。個人のレコードは、3 つの文字列データから成り立ちます。 **store** サブルーチンは以下のとおりです。

```
public sub store()
' store the data
  dim yn, id, rec, ln, fn
  input "Store? (y/n): ", yn ' see if user wants to store
  ' only go on if user says yes
  if ( yn <> "y" ) then
    println "...not stored."
    exit sub
  end if

  id = increment(^PersonD) ' generate a new id
  rec = name & "^" & phone & "^" & intdob ' concatenate data into a record
  ^PersonD( id ) = rec ' store the record
  ' ...remainder of store subroutine continued on next page...
end sub
```

ストレージ・サブルーチン(2)

このセクションでは、インデックスによって名前を姓と名に分割します。次の3文は、他言語の文とはまったく異なります。文のうち2つのデータは添え字に格納されるため、イコール記号の右側には“何も”指定しません。理由はツリー構造にあります。

例えば、姓、名、IDは添え字に格納されます。名前はアルファベット順にソートされるため、データベースには姓、名の順でインデックスを作成します。電話番号と誕生日のインデックスも同様に作成します。グローバルの`^PersonI`は3つの大きな枝に分かれ、各枝にインデックスがあるツリーになります。電話番号インデックスを構築する行は異なるため、電話番号の固有データを利用できません(IDは添え字に格納されません)。

複数のデータが入力されたと想定して配列のツリーを描画すると、配列のデータ構造が理解できます。これを実行するかどうかは別にして、システム管理ポータル([データ管理]セクション)を使用して[[ホーム]→[グローバル]]に移動し、2つのグローバル(`^PersonD`と`^PersonI`)を確認してください。

```
public sub store()
    ' ...store subroutine continued from previous page...
    ' break name for storage in index
    ln = piece(name, ",", 1) : fn = piece(name, ",", 2)

    ' the next three statements store data in subscripts
    ' because of the automatic sorting of subscripts,
    ' this has the effect of building 3 indexes: name, phone, and DOB
    ^PersonI("Name", ln, fn, id) = "" ' store the name
    ^PersonI("Phone", phone) = id    ' store the UNIQUE phone
    ^PersonI("DOB", intdob, id) = "" ' store the DOB
    println "...stored"              ' done
end sub
```

グローバルの例(1)

はじめに(G)
Cacheの開始(S)[CACHE200710369]
Cacheの停止(Q)
スタジオ(D)
ターミナル(T)
システム管理ポータル(P)
ドキュメント(N)
リモートシステム アクセス(R) ▶
優先接続サーバ(E)[CACHE200710369] ▶
Cacheバージョン情報(B)
終了(X)

ここでは、**myBASdatent** ルーチンに 3 項目を入力する例を示します。誕生日は内部形式で格納されます。グローバルを確認するには、システム管理ポータル ([データ管理]セクション)を使用して、SAMPLES ネームスペース内の[[ホーム]→[グローバル]]に移動します。 ^PersonD と ^PersonI のエントリを見つけ、それぞれに対して[[データ]]をクリックします。

^PersonI グローバルには、ソートされた 3 つのインデックスが含まれます。各インデックスの葉はソート順に並んでいます。DOB と Name インデックスは次元を追加して、同じ誕生日や名前を持つ個人データを複数格納できます。Phone インデックスは構造上、一意のデータのみ持ちます。

INTERSYSTEMS Global Data
 Licensed to: License missing or unreadable.
[Home](#) | [About](#) | [Help](#) | [Logout](#)
[\[Home\]](#) > [\[Globals\]](#) > [\[Global Data\]](#)

View global in namespace SAMPLES:

Global Search Mask:

Search History: ▼

```

^PersonD      = 3
^PersonD(1) = "Jones,Cleon^111-111-1111^37105"
^PersonD(2) = "Agee,Tommie^617-333-3333^37110"
^PersonD(3) = "Swoboda,Ron^222-222-2222^37779"
Total: 4      [End of global]

```

INTERSYSTEMS Global Data
 Licensed to: License missing or unreadable.
[Home](#) | [About](#) | [Help](#) | [Logout](#)
[\[Home\]](#) > [\[Globals\]](#) > [\[Global Data\]](#)

View global in namespace SAMPLES:

Global Search Mask:

Search History: ▼

```

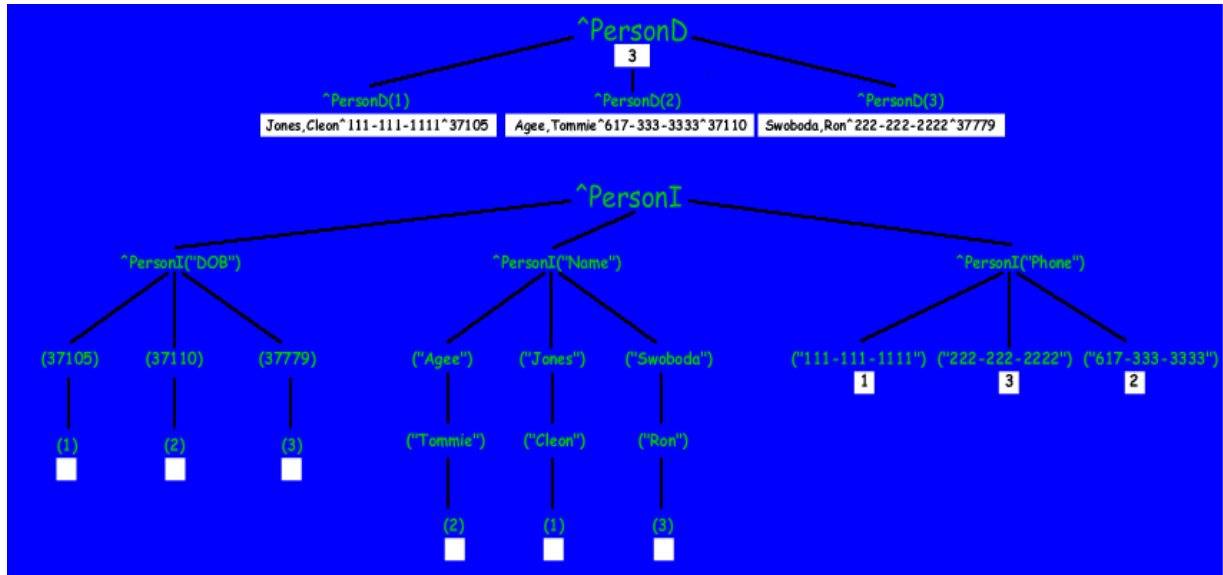
^PersonI("DOB",37105,1)      = ""
^PersonI("DOB",37110,2)     = ""
^PersonI("DOB",37779,3)     = ""
^PersonI("Name","Agee","Tommie",2) = ""
^PersonI("Name","Jones","Cleon",1) = ""
^PersonI("Name","Swoboda","Ron",3) = ""
^PersonI("Phone","111-111-1111") = 1
^PersonI("Phone","222-222-2222") = 3
^PersonI("Phone","617-333-3333") = 2
Total: 9                    [End of global]

```

次のページは、これらのグローバルのツリー・ダイアグラムを示しています。

グローバルの例(2)

以下の ^PersonI の図では、図を簡略化するために新規次元やレベルには追加された添え字のみを表示しています。



```
^PersonI("DOB",37105,1) = ^PersonI("DOB",37110,2) =
^PersonI("DOB",37779,3) =
```

```
^PersonI("Name","Agee","Tommie",2) =
^PersonI("Name","Jones","Cleon",1) =
^PersonI("Name","Swoboda","Ron",3) =
```

```
^PersonI("Phone","111-111-1111")=1 ^PersonI("Phone","222-222-2222")=3
^PersonI("Phone","617-333-3333")=2
```

演習 2

グローバル[^]PersonD(データ)とグローバル[^]PersonI(インデックス)の作用を理解する最適な方法は、各自が **myBASdatent** ルーチンで使用してみることです。

- 電話番号が電話インデックスに既に存在するかどうかをチェックするために、**validPhone()**関数を追加します。
- (Caché 演習ガイドから)ルーチンに **store** サブルーチンをコピーします。
- コンパイル後、ルーチンを実行し、個人データを格納します。システム管理ポータルを使用してグローバルを表示します。再度実行し、データをいくつか格納します。グローバル、特に[^]PersonI を再度チェックします。

詳細は、[Appendix E: 演習 2: ストレージ・サブルーチンの追加] を参照してください。

第 1 章の要約

第 1 章では、Cache Basic を強力にしている以下の機能について学習しました。

- 複数值文字列が持つ多くの機能
- Cache スタジオ・デバッガの使用法
- 配列、ツリー構造、グローバル変数の存在のテスト法
- データベースとインデックスの簡単な作成方法

第2章

データベースの使用法

第2章では、第1章で構築したデータベースをどのように使用するかを、Cache Basic の重要な機能と併せて学習します。この章の大部分の作業は、検索/編集ルーチンをインタラクティブに作成することです。

第2章では Basic の便利な機能について学習します。

- 関数: Traverse()、Lock()、Unlock()
- コマンド: TStart、TCommit
- 参照渡しのパラメータ
- New、OpenId と %Save の標準メソッドを使用して、Cache オブジェクトへアクセスするコードの記述方法

Traverse()関数(1)

第 1 章では、ほとんどのデータを添え字に格納する ^PersonI インデックスを作成しました。ここでは、添え字に格納されたデータにどのようにしてアクセスするかを学習します。配列にノードが存在する場合、**Traverse()**関数はその子ノードの添え字値にアクセスします。添え字は自動的にソートされるため、関数は子ノードを順番に走査します。

このためには、**Traverse** に 1 つ以上の添え字付き配列名を渡します。配列と最後に割り当てられた添え字以外の添え字は、ノードを識別します。最後の添え字は、**Traverse** 関数が使用します。x 配列を使用した例では、**Traverse** が集中した添え字が最初で唯一の添え字です。空文字列("")は配列の添え字には使用できませんが、**Traverse** に渡される配列の添え字として使用できます。これにより、**Traverse** は最初の有効な添え字値を返すことができます。

この例ではまず、**Traverse** は有効な最初の添え字 1 を返します。その後残りの添え字を順番に返すので、1 を渡すと 4 を、4 を渡すと 9 を返します。9 を渡すと **Traverse** はそれ以上有効な添え字がないことを示す空文字列を返します。

```
x(1)=1 : x(4)=2 : x(9)=3
```

```
println "first subscript is: ", traverse( x("") )
println "subscript after 1 is: ", traverse( x(1) )
println "subscript after 4 is: ", traverse( x(4) )
println "subscript after 9 is: ", traverse( x(9) )
```

Traverse()関数(2)

Traverse()を使用して有効な添え字値を取得する際に、同時に等式の右側の値も取得できます。**Traverse()**の 3 番目の引数に変数名を割り当てると、2 ステップの代わりに 1 ステップでこれらの値を取得できます。

```
x(1)=1 : x(4)=2 : x(9)=3
a = traverse( x(4) ) : b = x(a)
println a, " ", b ' get next subscript (a), and value on right (b)
a = traverse( x(4), 1, b)
println a, " ", b ' do the same thing, simpler and faster
```

View global in namespace SAMPLES:

Global Search Mask:

Search History: ▼

^PersonI("DOB",37105,1)	=	""
^PersonI("DOB",37110,2)	=	""
^PersonI("DOB",37779,3)	=	""
^PersonI("Name","Agee","Tommie",2)	=	""
^PersonI("Name","Jones","Cleon",1)	=	""
^PersonI("Name","Swoboda","Ron",3)	=	""
^PersonI("Phone","111-111-1111")	=	1
^PersonI("Phone","222-222-2222")	=	3
^PersonI("Phone","617-333-3333")	=	2
Total: 9		[End of global]

ここで、もう一度 `^PersonI` と `Traverse()` を確認します。 `^PersonI` には添え字を 4 つまで設定できますが、この例では、ルートの子、つまり第 1 次元を検索します。 `Traverse()` 関数は最初の有効な添え字 "DOB" を返します。 "DOB" を渡すと、次に存在する添え字 "Name" を返します。 "Name" を渡すと "Phone" を返し、 "Phone" を渡すと `Traverse()` は空文字列を返します。 `Traverse()` でこのレベルを検索すると、各添え字の 1 つを返します。 ここに示すビューでは、 "DOB"、 "Name" および "Phone" の値を 3 回ずつ繰り返します。 これはエラーではなく、ポータルのグローバルの表示方法が異なるためです。 配列ツリーでは、 "DOB"、 "Name"、 "Phone" はそれぞれ 1 つのノードで構成されているからです。

Note:

このページ以降 5 ページに渡る `Traverse()` 例は、演習での指定により、SAMPLES ネームスペースのデータベースに少なくとも 1 人分のデータを格納してからのみ実行できます。

```
println "first node of ^PersonI is: ", traverse( ^PersonI( "" ) )
println "node after ""DOB"" is: ", traverse( ^PersonI( "DOB" ) )
println "node after ""Name"" is: ", traverse( ^PersonI( "Name" ) )
println "node after ""Phone"" is: ", traverse( ^PersonI( "Phone" ) )
```

Traverse ループ



View global in namespace SAMPLES:

Global Search Mask:

Search History:

^PersonI("DOB",37105,1)	=	""
^PersonI("DOB",37110,2)	=	""
^PersonI("DOB",37779,3)	=	""
^PersonI("Name","Agee","Tommie",2)	=	""
^PersonI("Name","Jones","Cleon",1)	=	""
^PersonI("Name","Swoboda","Ron",3)	=	""
^PersonI("Phone","111-111-1111")	=	1
^PersonI("Phone","222-222-2222")	=	3
^PersonI("Phone","617-333-3333")	=	2
Total: 9		[End of global]

前ページの例では、**Traverse()** を繰り返し使用してツリーの検索を実行しました。ループを使用するとより簡単に取得でき、ツリーの他のレベルの検索もできます。以下の例は、グローバルで頻繁に使用するコードで、学習すると役立ちます。

最初に、変数 *ln* を初期化して空文字列にします。その後 **Do/Loop** ループを開始します。割り当て文では、**Traverse()** を使用して `^PersonI("Name")` ノードで最初の子、つまり姓の添え字値を生成します。次に、**If** コマンドは、ループを終了させるかどうかを判断し、空文字と等しい姓を検索します。最後に、値に対してどのような処理を実行するかを指定します。その後、ループは同じ処理を繰り返し、次の姓を検索します。電話番号は姓とは兄弟でないため、姓の“隣に”電話番号の添え字が存在してもその値は取得しません。

Traverse() を使用する割り当て文は、典型的なプログラミング・コード `i = i + 1` と類似しています。このコードは、変数値の現在の値を使用して新規の値を生成します。

```
ln = ""
do
  ln = traverse(^PersonI("Name", ln))
  if ln = "" then exit do
  println ln
loop
```

コード例は、*BAStraverse.BAS* の **simpleloop** サブルーチンにあります。Cache ターミナルを使用して実行します。

```
SAMPLES>do simpleloop^BAStraverse()
```

入れ子になった Traverse ループ

入れ子になった **Traverse()** ループ(ツリーの枝内の各レベルに 1 つ)を使用すると、ID 番号を示す葉ノードに到達できます。その後 ID 番号を使用してデータ配列に戻り、必要な情報を取得します。例として、名前インデックスを使用します。ここでは、各姓データに対応する名前すべてを検索するために、新規のループを呼んでいます。また、そのフルネームに対し、新規ループで該当するすべての ID 番号を検索します。ID を使用して、**^PersonD** の残りのデータを取得します。

```

In = ""
do
  In = traverse(^PersonI("Name", In) )
  if In = "" then exit do
  fn = ""
  do
    fn = traverse(^PersonI("Name", In, fn) )
    if fn = "" then exit do
    id = ""
    do
      id = traverse(^PersonI("Name", In, fn, id) )
      if id = "" then exit do
      rec = ^PersonD(id)
      print piece(rec, "^", 1), space(2),
      print piece(rec, "^", 2), space(2)
      println DateConvert(piece(rec, "^", 3), vbToExternal)
    loop
  loop
loop

```

コード例は、**BAStraverse.BAS** の **nameloop** サブルーチンにあります。Cache ターミナルを使用して実行します。

```
SAMPLES>do nameloop^BAStraverse()
```

Traverse ループ

`Traverse()`ループで `ln = ""` は、“姓の最初から開始”を意味します。同様に、`if ln = "" then exit do` は、“姓の最後で停止”を意味します。文を修正してループを使用することにより、生成される添え字を少なくします。例えば、`ln = "B"` と記述すると、ループが “Agee” をスキップします。

このループで名前の部分文字列から開始する名前のみを生成する場合、変数 `ln` を開始する部分文字列と等しい値に設定します。また、`If` 文を変更して、部分文字列に続く文字で始まる名前に到達する前に、ループを停止させる必要があります。これにはいくつか方法があります。内容については次ページで学習します。

```
ln = ""
do
  ln = traverse(^Personl("Name", ln) )
  if ln = "" then exit do
  println ln
loop
```

Traverse ループの開始

ここでは、ループの開始について学習します。変数 `substr` は、空でない部分文字列を格納すると想定します。先述の例のとおり、`ln` を "" に設定する代わりに、以下の文 (`ln = substr`) を使用して、部分文字列を元に **Traverse()** ループを開始します。しかし、毎回 `ln` に部分文字列と等しい値を設定すると、検出されないデータが発生します。`substr` がデータベース内の姓と等しい場合はどうなるでしょうか。このような場合、ループは次の名前から開始されます。これは、初期 **Traverse()** が原因で、インデックスに `substr` と完全に一致する名前が存在しても同様です。対処法として、**Traverse()** 文を **Do/Loop** 文の最後に配置します。最初に **Exists()** を使用して、`substr` がデータベースに存在する姓に一致するかどうかを検証します。一致しない場合、**Traverse()** は `substr` に一致する可能性のある最初の値に移動します。その後ループは処理を継続し、`substr` に続く名前を生成します。しかし、ループの終了に焦点を置いて、“Swoboda”の値を取得しないようにする必要があります。

```
substr = "J"
if not exists( ^Personl("Name", substr)) then
    ln = traverse( ^Personl("Name", substr))
end if
do
    if ln = "" then exit do
    println ln
    ln = traverse( ^Personl("Name", ln) )
loop
```

コード例は、`BAStraverse.BAS` の **loopstart** サブルーチンにあります。Caché ターミナルを使用して、`substr` を渡すコードを実行します。

```
SAMPLES>do loopstart^BAStraverse("J")
SAMPLES>do loopstart^BAStraverse("Jones")
```

Traverse ループの終了

ここでは、ループの終了方法について学習します。ここで再度、変数 `substr` には空文字以外の部分文字列が設定されているとします。ここで、`(if left(ln ,len(sub)) <> substr then exit do)` は、“ln の開始が `substr` と一致しない場合、ループを終了する”ことを意味します。また、`ln = ""` がある名前の最後に `Traverse()` が到達する場合も処理します。つまり、ln が空文字列の場合は ln の `Mid()` が空文字列で、`substr` は空文字列でないため、ループが終了します。以下にいくつか例を挙げます。

```
println "first test: "
substr = "J" : ln = "Jones"
if left(ln, len(substr)) <> substr then println "No Match"
println "second test: "
substr = "J" : ln = "Swoboda"
if left(ln, len(substr)) <> substr then println "No Match"
println "third test: "
substr = "J" : ln = ""
if left(ln, len(substr)) <> substr then println "No Match"
substr = "J"
if not exists( ^Personl("Name", substr)) then
    ln = traverse( ^Personl("Name", substr))
end if
do
    if left(ln, len(substr)) <> substr then exit do
    println ln
    ln = traverse( ^Personl("Name", ln) )
loop
```

コード例は、`BAStraverse.BAS` の `loopend` サブルーチンにあります。Caché ターミナルを使用して、`substr` を渡すコードを実行します。

```
SAMPLES>do loopend^BAStraverse("J")
SAMPLES>do loopend^BAStraverse("Jones")
```

別のルーチンの呼び出し

次の演習を始める前に、モジュールのサブルーチンを他のモジュールから呼び出す構文を学習する必要があります。これまで、Caché ターミナルからの Basic サブルーチン呼び出しは、サブルーチン名とルーチン名の間に “^” を置く ObjectScript 構文 `do main^myBASdatent()` を使用していました。Basic あるいは ObjectScript のいずれかのサブルーチンを Basic から呼び出すには、サブルーチン名とルーチン名の間に “@” を置く Basic 構文 `main@myBASdatent()` を使用します。

サブルーチンに引数を渡す場合、Basic と ObjectScript は異なる構文を使用します。

渡す引数

呼び出し元言語	呼び出し先言語	構文	例
ObjectScript	ObjectScript	既定は値渡しです。呼び出しコードは、先行するピリオド (.) 付きの引数で、どの引数が参照渡しされるかを指定します。	<code>do sub^rou(a, .b)</code>
ObjectScript	Basic	既定は値渡しです。呼び出しコードは、先行するピリオド (.) 付きの引数で、どの引数が参照渡しされるかを指定します。呼び出されたメソッドのシグニチャは、 ByVal あるいは ByRef 引数修飾子を使用する場合がありますが、影響はありません。	<code>do sub^rou(a, .b)</code>
Basic	ObjectScript	既定は参照渡しです。呼び出しコードは、先行する ByVal 付きの引数で、どの引数が値渡しされるかを指定します。呼び出しコードは、明確にするために ByRef も使用できます。	<code>sub@rou(ByVal a, b)</code>
Basic	Basic	この呼び出しは、呼び出されたメソッドのシグニチャに依存します。呼び出しコードは、 ByRef あるいは ByVal を使用したシグニチャを上書きできます。	<code>sub@rou(a, b)</code>

演習 3

ここでは lookup ルーチンを開始します。このルーチンは、名前、電話番号、誕生日を検索します。以下に、記述が必要なサブルーチンの名前と用途を示しています。

- **myBASlookup** を呼び出し、新規の Basic ルーチンを生成します。最初に **main** セクションがあり、**getsub mit**、**help**、**getdob**、**getphone**、**getname**、**display**、**pick** のサブルーチンを含みます。
- **main** セクションには **getsubmit** を呼び出すループがあり、ユーザに文字入力を要求します。フルネーム/名前の一部、電話番号/電話番号の一部、誕生日を入力します。 ? を入力するとヘルプを表示し、Enter を押すと終了します。 **getsubmit** で **validDOB@myBASdatent()** を使用して誕生日を検証します。ユーザが ? を入力すると、**help** サブルーチンは lookup の代替案をリストします。
- **getsubmit** でユーザが有効な誕生日を入力すると、**getdob** サブルーチンは誕生日の内部形式が *^PersonI* に存在するかどうかをチェックし、インデックスをループして、同じ誕生日のデータの個人 ID すべてを検出します。
- **getsubmit** でユーザが電話番号の全部あるいは一部を入力すると (617、617-621、または 617-621-0600)、**getphone** サブルーチンは一致するデータを検索します。 **Traverse()** の 3 つの引数を指定します。メモ : **getphone** を記述しているときに、バグが偶然発生する可能性があります。解決方法は、演習ガイドを参照してください。
- **getsubmit** でユーザがフルネームあるいは名前の一部 (Smith,John、Smith, J、Sm, John、Sm, J) を入力すると、**getname** サブルーチンは一致するデータを検索します。 **formatName@myBASdatent()** を使用して検索前に名前をフォーマットします。
- 一致したデータ・リストに対し、**display** サブルーチンで個人データを表示します。パラメータとして ID を受け入れ、個人データをすべて表示します。**display** は、“line” あるいは “table” のいずれかの別の引数も受け入れます。**display@myBASdatent()** を使用して、“line” を選択した場合は一致した値を横並びのリスト形式で表示し、“table” を選択した場合はユーザの選択を縦に表示します。
- ローカル配列を使用して一致する値を追跡し、表示された一致番号に連結する ID 番号を取得します。**pick** サブルーチンを記述し、一致する値をユーザが選択できるようにします。

詳細は、[Appendix F: 演習 3: LookUp ルーチンの記述] を参照してください。

データベースの整合性

Basicには、データベースの整合性を確実にする関数、**Lock()**や**Unlock()**と、コマンド**TStart**と**TCommit**があります。**myBASlookup** ルーチンに編集機能を追加するために、これらの関数とコマンドを学習します。コマンドは、通常組み合わせて使用されます。以下はコマンドの一般的な使用方法です。

1. 修正するレコードを選択
2. レコードを **Lock()**
3. 更新データを取得
4. **TStart** を実行
5. 新規データを格納
6. トランザクションに **TCommit** を指定
7. レコードのロックを **Unlock()**

Lock()コマンドを使用すると、複数のユーザが同じレコードを同時に更新しようとする競合プロセスを防ぐことができます。ただし、規約に従った場合にのみ機能します。任意のグローバルを更新するアプリケーションのコードはすべて、更新されるレコードを必ず **Lock()** するよう試みる必要があります。終了後は **Unlock()** する必要があります。ルーチンによって **Lock()** を使用しているものとしていないものがある場合は、前者のルーチンがレコードをロックするのに対し、後者のルーチンはレコードを更新します。

TStart コマンドと **TCommit** コマンドを使用すると、データベース・トランザクションを保護できます。トランザクションは、データベースに対する一連の変更処理(挿入、更新、削除)です。すべてのトランザクションを完了させるか、あるいはまったく実行しないかのどちらかにする必要があります。これにより、トランザクションのグローバルすべて(例えばデータとインデックスの両方)が同期の状態を保ちます。

Lock()関数とUnlock()関数(1)



Remove All Locks

Following items are currently locked on your system:

[Last](#)

Owner	Mode/Count	Reference	Directory
3024	Exclusive	^%SYS("CSP","Daemon")	c:\cachesys\mgr\
3024	Exclusive	^%SYS("CSP","Reclaim")	c:\cachesys\mgr\
<u>3068</u>	Exclusive	^TASKMGR	c:\cachesys\mgr\
<u>188</u>	Exclusive	^x(1)	c:\cachesys\mgr\samples\

ここでは **Lock()** および **Unlock()** の機能と使用方法について学習します。ロックする場合、システムはロック・テーブルに **Lock()** コマンドの引数を追加します。別のプロセスで同じレコードをロックしようとする、最初のロックが解除されるまでそのプロセスを待機します。2番目のプロセスが異なるレコードを編集し、レコードを **Lock()** すると、システムはロック・テーブルにもそのレコードを追加します。処理がうまくいくと **Lock()** は真を返しますが、それ以外の場合は偽を返します。

Lock() の 2 番目の引数は、タイムアウトの秒です。処理が指定時間内に正常終了しない場合、**Lock()** は偽を返します。

以下の例を実行する前に、システム管理ポータル([操作]セクション)を使用して [[ホーム]→[ロック]]に移動し、ロック・テーブルを表示します。例を実行すると、完了するまで数秒かかるためロックを表示できます。[ロック・テーブル]フォルダをクリックすると、表示を更新できます。

```
if lock(^x(1), 3) then
    println "lock successful"
    ' wait several seconds so lock remains in Lock Table
    for i = 1 to 100000000 : next i
else
    println "lock failed"
end if
```

ロックが失敗した場合、Cache ターミナルと ObjectScript **Lock** コマンドを使用して、同じ項目をロック・テーブルに追加します。その後上記コードを実行します。

```
SAMPLES>lock ^x(1) ; add ^x(1) to the Lock Table before running example
```

```
SAMPLES>lock          ; to remove ^x(1) from the Lock Table afterwards
```

myBASlookup ルーチンの次の段階では、ユーザが個人データを修正できるコードを記述します。つまり、`^PersonD` グローバルと `^PersonI` グローバルを更新します。この場合、ユーザが更新するレコードと特定の ID をいったん選択すると、これをルーチン `lock + ^PersonD(id)` へ追加します。新規データを格納した後、`unlock(^PersonD(id))` を使用してレコードのロックを解除します。

Lock()関数とUnlock()関数(2)

すべての **Lock()** コマンドは、ロック・テーブルのエントリを確認、追加、削除するだけなので、**Lock()** を使用する場合は、既存グローバルの名前を指定する必要はありません。例えば、`^PersonD` を編集する際は常に、`^Person` が存在しなくても、`(lock(^PersonD(id)))` の代わりに `lock(^Person(id))` を実行できます。この場合は `^Person` がロック・テーブルに追加されるので、同様に機能します。また、`^PersonD` 配列を編集するすべてのコードが `^Person` をロックします。

このグローバルには個人データが含まれますが、個人データを編集する際に `^PersonI` をロックする必要はありません。これは、実際に配列を“ロックしている”わけではなく、単にエントリをロック・テーブルに追加しているだけで、グローバルの編集を許可するコードをロックするためです。アプリケーションでは、`^PersonD` を更新するすべてのコードが、同じ規約を使用します。

`^PersonD` を更新するコードはすべて必ず同時に `^PersonI` も更新するため、`lock(^PersonD(id))` だけで十分です。

`^PersonD(1)` がロック・テーブルにある場合は、別のプロセスで `^PersonD(1)` をロックすることはできません。ただし、`^PersonD(2)` はロックできます。つまり、ロック・テーブルには別のプロセスであれば兄弟のエントリを追加できます。しかし、別のプロセスはロック・テーブルの祖先のエントリをロックできません(この場合は `^PersonD`)。これは、下位層のノードがロックされた場合は、その上位層の配列にもアクセスさせないためです。同様に、別のプロセスはロック・テーブルのエントリの子孫もロックできません (例: `^PersonD(1,2)`)。ノード自身がロックされた場合は、そのノードの子孫にもアクセスさせないためです。

トランザクション処理

データの保存/削除をする場合、**TStart** コマンドを文の前に置き、その後に **TCommit** コマンドを置きます。これにより、文はトランザクションとして処理されます。システムがトランザクション中にクラッシュした場合、(操作が復元した際に)不完全なトランザクションをロールバック(あるいはキャンセル)します。すべてのトランザクションが完了しているか何も実行されていないかのどちらかになります。

次のセクションで学習しますが、Caché オブジェクトや SQL テーブルでのオペレーションの保存や削除は、トランザクション内で自動的に実行されます。

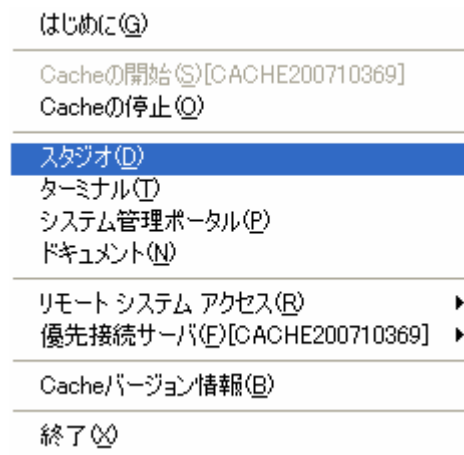
演習 4

myBASlookup ルーチンに編集機能を追加します。

- ユーザがリストから一致項目を選択すると、**main** にコードを追加し、新規サブルーチン **edit** を呼び出します。
- **edit** サブルーチンは、編集のために選択されたレコードをロックします。その後、**load**、**reprompt**、**update** を呼び出します。更新終了後、レコードのロックを解除します。
- **load** サブルーチンは個人データをロードします。**display** サブルーチンを更新して、新規サブルーチンを呼び出します。
- **reprompt** は関数です。**myBASdatent** プロンプトに類似し、“有効な”関数を使用しますが、プロンプト内部の現在の(古い)データも表示します。変更された場合、真を返します。
- 値が変更された場合、**update** サブルーチンがトランザクションを開始し、^PersonD に新規データを保存します。次に、^PersonI を更新します。インデックスの更新とは、つまり、インデックスの古いエントリで **Erase()** を使用して、新規のエントリを追加することです。

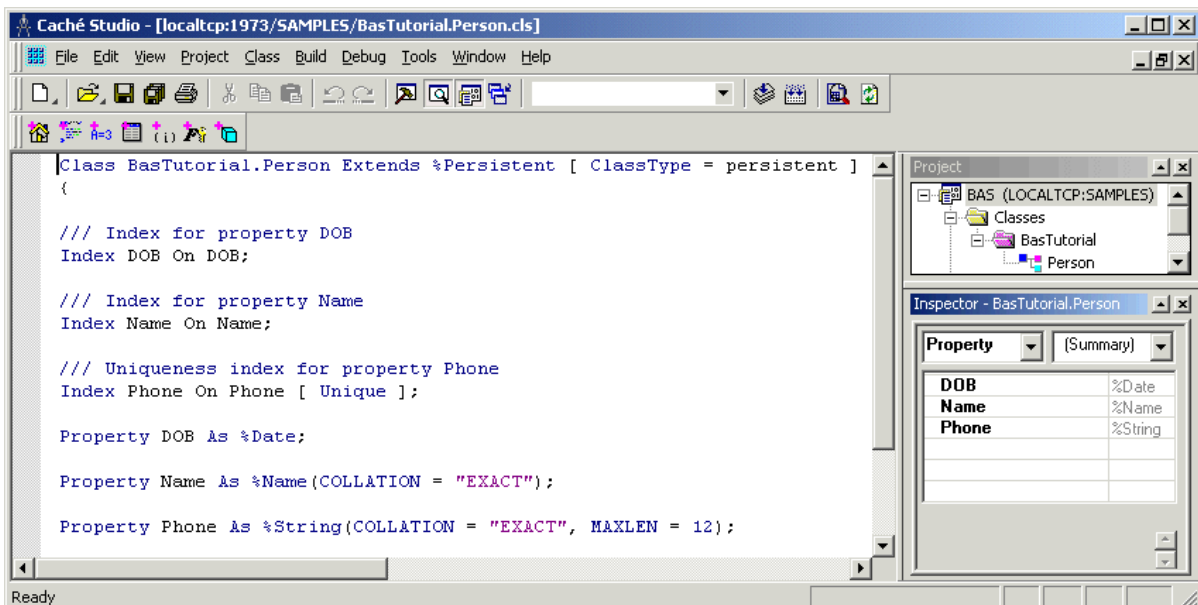
詳細は[Appendix G: 演習 4: 編集機能の追加] を参照してください。

Cache オブジェクト



Basic もオブジェクトにアクセスするコードを記述できます。このチュートリアルでは、Cache のクラス、オブジェクト、プロパティ、メソッド、クエリについての説明は省略しますが、これらのトピックに関して学ぶには、1 つ以上の他のチュートリアルを試してください。ここでは、Persons データベースを使用して学習します。 ^PersonD グローバルのエントリは、1) **Person** クラスの永続オブジェクトや、2) Person リレーショナル・テーブルの行と考えることができます。Cache オブジェクトを簡単に説明します。

Cache スタジオを使用して、SAMPLES から BAS プロジェクトをリロードします。**[Classes]**フォルダの**[BasTutorial]**パッケージにある **Person** クラスには、3 つのプロパティ、Name、Phone、DOB があります。このクラスによって、さらに簡単な構文を使用しての Person オブジェクトの利用が可能となります。区切り文字を考慮する必要はありません。



Caché メソッド

Person クラスは組み込みのメソッドも提供しています。メソッドとはオブジェクトで実行する Basic サブルーチンのことです。使用できるメソッドは以下のとおりです。

- **New** はローカル・メモリで新規の Person オブジェクトを生成します。
- **OpenId** は永続 Person オブジェクトをローカル・メモリにロードします。**OpenId** は自動的にロックできます。
- **%Save** はローカル・メモリでの Person オブジェクトのデータを確認します。正しく確認された場合は、そのデータを保存します。オブジェクトの**%Save** は、トランザクションとして自動的に作動します。

新規に Person オブジェクトを作成する場合は、**New** を使用してデータを設定し(以下に示している“ドット構文”を使用します)、**%Save** を実行します。既存オブジェクトを編集するには、**OpenId** を使用してデータを更新し(ここでもドット構文を使用します)、**%Save** を実行します。

Person オブジェクトのプロパティを参照するには、ドット構文(オブジェクト・プロパティ)を使用します。下記 1 行目は ID が 2 の既存の Person オブジェクトを開きます。次に、データの保存方法に関係なく、**per** を使用して person オブジェクトのプロパティを参照できます。**per** を Nothing に設定して、person オブジェクトのローカル・コピーを削除します。

Note:

この例は、次ページで説明しているように、データベースを変換した後の実データを示しています。

```
per = OpenId BasTutorial. Person (2)
println "Name: ", per. Name
println "Phone: ", per. Phone
println "DOB: ", DateConvert(per. DOB, vbToExternal)
per = Nothing
```

Caché オブジェクトストレージ

Cachéクラスは、既定では区切り文字列の代わりにリストを使用し、構築したグローバルを使用します。チュートリアルを続けるにあたり、`^PersonD`をこの配列構造を使用するように変換し、また`^PersonI`の“Phone”セクションに小規模な変更を加えます。**BASdbconvert**と呼ばれる、既に記述されたルーチンがあります。Cachéターミナル・セッションを開始し、`do main ^BASdbconvert ()`を実行します。

変換後は、**myBASdatent**と**myBASlookup**ルーチンは作動しません(次の演習では、新規のバージョンを作成します)。システム管理ポータルで変換の結果を参照できます。ここで各レコードは、3つの要素をリスト表示(**\$ListBuild**関数により)しています。また、`^PersonI`の“Phone”セクションが若干変更され、IDが添え字になっています。

データを元の形式に変換する場合は、`do goback ^BASdbconvert ()`を実行してください。その結果、**myBASdatent**と**myBASlookup**を再度使用することができ、自由に行き来することができます。

INTERSYSTEMS Global Data
Licensed to: License missing or unreadable.
Home | About | Help | Logout
[Home] > [Globals] > [Global Data]

View global in namespace SAMPLES:

Global Search Mask:

Search History:

```

^PersonD      = 3
^PersonD(1) = $lb("Jones,Cleon","111-111-1111","37105")
^PersonD(2) = $lb("Agee,Tommie","617-333-3333","37110")
^PersonD(3) = $lb("Swoboda,Ron","222-222-2222","37779")
Total: 4      [End of global]
    
```

INTERSYSTEMS Global Data
Licensed to: License missing or unreadable.
Home | About | Help | Logout
[Home] > [Globals] > [Global Data]

View global in namespace SAMPLES:

Global Search Mask:

Search History:

```

^PersonI("Phone","111-111-1111",1) = ""
^PersonI("Phone","222-222-2222",3) = ""
^PersonI("Phone","617-333-3333",2) = ""
Total: 3      [End of global]
    
```

演習 5

myBASdatent と **myBASlookup** の新規バージョン、**myBASdatentobj** と **myBASlookupobj** を生成します。

- Caché スタジオを使用して **myBASdatent** ルーチンをロードし、**myBASdatentobj** として保存します。データを格納するオブジェクト構文を使用するため **store** サブルーチンを編集します。
- **myBASlookup** ルーチンをロードし、**myBASlookupobj** として保存します。**load** サブルーチンを編集し、オブジェクト構文で ^PersonD から選択した個人がロードされるようにします。**update** サブルーチンを編集して、オブジェクト構文で ^PersonD にデータを格納するようにします。^PersonI の新規構造が反映されるように **phone** サブルーチンを編集します。

詳細は、[Appendix H: 演習 5: データ入力と検索のオブジェクト・バージョンの記述] を参照してください。

第 2 章の要約

第 2 章では Basic の便利な機能について学習しました。

- 関数: Traverse()、Lock()、Unlock()
- コマンド: TStart、TCommit
- 参照渡しのパラメータ
- New、OpenId と %Save の標準メソッドを使用して、Cache オブジェクトへアクセスするコードの記述方法

このチュートリアルをご利用いただき、ありがとうございました。

Cache Basicについてさらに学習するには、[Cacheドキュメント: \[Cache開発リファレンス\]-\[Cache Basicリファレンス\]](#) を参照してください。

プロジェクトのすべてを 1 つのポータブル・ファイルに保存する方法は以下のとおりです。

1. **[スタジオ]**を起動します。
2. **[ファイル]→[ネームスペース変更]**をクリックして、[SAMPLES] ネームスペースに接続します。
3. **[ツール]→[エクスポート]**をクリックします。
4. **[現プロジェクトのエクスポート]**と**[ローカル・ファイルにエクスポート]**オプションを選択します。
5. フォルダを調べ、mybas.xml を指定して、**[保存]**をクリックします。
6. **[OK]**ボタンをクリックします。

Appendix B:

サンプル・アプリケーション

Caché をインストールすると、*SAMPLES* ネームスペースで、チュートリアル・アプリケーションの開発や実行に必要なすべてのファイルもインストールされます。

BasTutorial.xml ファイルは `c:¥cachesys¥dev¥tutorials` にインストールされます。

アプリケーションを *SAMPLES* に再ロードしたり、他のネームスペースにロードする場合、以下の作業を行います。

1. **[スタジオ]** を起動します。
2. **[ファイル]**→**[ネームスペース変更]** をクリックし、アプリケーションをロードするネームスペースに接続します。
3. **[ツール]**→**[ローカルからインポート]** をクリックし、**[XML]** を選択します。
`c:¥cachesys¥dev¥tutorials` を検索します。
4. *[BasTutorial.xml]* ファイルを選択し、**[開く]** をクリックします。

Appendix C:

Right Triangle ルーチン

```

' RightTriangle compute area and hypotenuse of a right triangle
' this routine contains examples of Cache Basic features */

Sub Run()
println "Compute the area and hypotenuse of a right triangle"
println "given the lengths of its two sides."
println
println "First, choose a unit of measurement. "
input "(i)nches, (f)eeet, (m)iles, " _
      , "(c)entimeters, m(e)ters, (k)ilometers: ", units
println
' translate units to a full word
select case left(units, 1)
  case "i" units = "inches"
  case "f" units = "feet"
  case "m" units = "miles"
  case "c" units = "centimeters"
  case "e" units = "meters"
  case "k" units = "kilometers"
  case else units = "units"
end select

do
  println
  input "Length of side 1: ", side1
  if (side1) = "" then exit do
loop while IsNegative( side1 )
if (side1 = "") then exit sub

do
  println
  input "Length of side 2: ", side2
  if (side2) = "" then exit do
loop while IsNegative( side2 )
if (side2 = "") then exit sub

Compute(units, side1, side2)
end sub

```

```
public function IsNegative(ByVal num As %String) As %Boolean
' is num negative?
' check in range "1" through "9"
if (num < chr(49)) or (num > chr(57)) then
    print " Enter a positive number."
    return True
else
    print " Accepted."
    return False
end if
end function

private function Compute(ByVal units As %String, _
                        ByVal A As %Integer, _
                        ByVal B As %Integer)
' compute and display area and hypotenuse

area = round((( A * B ) / 2), 2)
hypot = round(sqr(( A ^ 2 ) + ( B ^ 2 )), 2)

println : println
println "The area of this triangle is ", area, " square ", units, "."
println
println "The hypotenuse is ", hypot, " ", units, "."
end function
```

Appendix D:

演習 1 : データ入力ルーチンの変更

1. **validName** 関数を記述します。

```
public function validName(name As %String) As %String
' validate a name - just checks for 2 pieces using ", "
' returns 0 for an invalid name and writes error message
' else returns formatted name
if len(name, ",") = 2 then
    return formatName(name)
else
    print "Enter Last,First" : println
    return 0
end if
end function
```

2. **formatName()** 関数を記述します。

```
public function formatName(name As %String) As %String
' change user's entry into proper name format
' SMITH, JOHN and smith, john -> Smith, John
dim ln, fn
ln = piece(name, ", ", 1) : fn = piece(name, ", ", 2)
ln = ucase(left(ln, 1)) & lcase(mid(ln, 2, len(ln)))
fn = ucase(left(fn, 1)) & lcase(mid(fn, 2, len(fn)))
return ln & ", " & fn
end function
```

3. **validPhone** 関数を記述します。電話番号の検証後、必要な場合は既定エリア・コードを追加します。

```
public function validPhone(phone As %String) As %String
' validate a phone - just checks for 3 pieces using "-" and length
' returns 0 for an invalid phone and writes error message
' else returns unchanged phone with default area code added
if (len(phone) = 8 and len(phone, "-") = 2) then
    phone = "617-" & phone ' add default area code
end if

if (len(phone) = 12 and len(phone, "-") = 3) then
    return phone
else
    print "Enter ###-###-#### or ###-####" : println
    return 0
end if
end function
```

4. **validDOB** 関数を記述します。無効データには、**On Error Goto** を使用する必要があります。未来の日付を許可しないようにします。

```
public function validDOB(dob As %String) As %Integer
' validate a date of birth – a valid date before or equal to today
' returns 0 for invalid dates and writes error message
' else returns internal format for valid dates
On Error Goto BadDate
    if DateDiff("d", dob, Date) < 0 then
        print "Enter a date in the past" : println
        return 0
    else
        return DateConvert(dob, vbToInternal)
    end if
BadDate:
    print "Invalid date" : println
    return 0
end function
```

5. **prompt** サブルーチンを編集します。**Do/Loop While** 文で、新規の関数 "\$\$valid" を 3 つ使用し、データが正しく入力されるまでプロンプトを繰り返します。

```
private sub prompt()
' subroutine for prompting
do
    input "Name: ", name : println
    if (name = "") then exit sub
    name = validName(name)
loop while name = 0
do
    input "Phone (617): ", phone : println
    phone = validPhone(phone)
loop while phone = 0
do
    input "DOB: ", dob : println
    intdob = validDOB(dob)
loop while intdob = 0
println
end sub
```

6. **display** サブルーチンを編集します。外部形式にデータを変換するコードを追加します。

```
public sub display()  
    ' subroutine for displaying data  
    println "Name:", space(15), name  
    println "Phone:", space(14), phone  
    println "DOB:", space(16), DateConvert(intdob, vbToExternal)  
    println  
end sub
```

Appendix E:

演習 2 : ストレージ・サブルーチンの追加

1. Caché スタジオを開始し、**myBASdatent** ルーチンを開きます。終了したルーチンも、*BASdatent2.BAS* として *SAMPLES* ネームスペースで使用できます。
2. **validPhone** 関数に **If** 文を追加し、電話番号データがインデックス内に既に存在するかどうかを確認します。

```
public function validPhone(phone As %String) As %String
'validate a phone - just checks for 3 pieces using "-" and length
'returns 0 for an invalid phone and writes error message
'else returns unchanged phone with default area code added
  if (len(phone) = 8 and len(phone, "-") = 2) then
    phone = "617-" & phone ' add default area code
  end if

  if (len(phone) = 12 and len(phone, "-") = 3) then
    if exists(^PersonI("Phone", phone)) then
      print "Phone number in use" : println
      return 0
    else
      return phone
    end if
  else
    print "###-###-#### or ###-####" : println
    return 0
  end if
end function
```

3. 新規の **store** サブルーチンを追加します。

```

public sub store()
' store the data
  dim yn, id, rec, ln, fn
  input "Store? (y/n): ", yn ' see if user wants to store
' only go on if user says yes
  if ( yn <> "y" ) then
    println "...not stored."
    exit sub
  end if

  id = increment(^PersonD) ' generate a new id
  rec = name & "^" & phone & "^" & intdob ' concatenate data into a record
  ^PersonD( id ) = rec ' store the record
' break name for storage in index
  ln = piece(name, ",", 1) : fn = piece(name, ",", 2)

' the next three statements store data in subscripts
' because of the automatic sorting of subscripts,
' this has the effect of building 3 indexes: name, phone, and DOB
  ^PersonI( "Name", ln, fn, id) = "" ' store the name
  ^PersonI( "Phone", phone) = id ' store the UNIQUE phone
  ^PersonI( "DOB", intdob, id) = "" ' store the DOB
  println "...stored" ' done
end sub

```

4. データを複数格納し、システム管理ポータルでグローバルを表示します。

Appendix F:

演習 3 : Lookup ルーチンの記述

1. Caché スタジオを開始し、新規ルーチンを作成します。このルーチンも BASlookup1.BAS として、SAMPLES ネームスペースで使用できます。
2. 以下は、**main** サブルーチンです。

```
Option Explicit
dim id, name, phone, intdob, matches

public sub main()
  dim done
  do
    getsubmit(id, done) ' let user submit a string for lookup
    if id = 0 then continue do
    display(id, "table") ' display the chosen person
  loop until done
end sub
```

3. 以下は、**getsubmit** サブルーチンです。

```
private sub getsubmit(ByRef id as %Integer, ByRef done as %Boolean)
' ask user what to search for, and take appropriate action
  dim submit
  id = 0 : done = False
  println : input "Lookup: ", submit : println
  if (submit = "") then ' user entered nothing
    done = True
    exit sub
  end if
' figure out what user entered
  if (submit = "?") then ' display help
    help()
  elseif (InStr(submit, "--") or (submit > 0 and submit < 999)) then
    println "...finding phone number"
    getphone(submit, id)
  elseif InStr(submit, ",") then
    submit = formatName@myBASdatent(submit)
    println "...finding name"
    getname(submit, id)
  elseif validDOB@myBASdatent(submit) then
    println "...finding birthday"
    getdob(submit, id)
' else it's an error
  end if
end sub
```

4. 以下は、**help** サブルーチンです。

```
private sub help()
' display different types of lookups
  println "You can enter:"
  println space(10), "* full name: Smith, John"
  println space(10), "* last name only: Smith,"
  println space(10), "* partial name: Sm, J or Smith, J or Sm, John"
  println space(10), "* phone number with area code: 617-621-0600"
  println space(10), "* partial phone numbers: 617 or 617-621"
  println space(10), "* date of birth"
  println
end sub
```

5. 以下は、**getdob** サブルーチンです。

```
private sub getdob(dob as %String, ByRef id As %Integer)
' perform dob lookup
' no partial matches
' if user picks a name from the list, id is returned to the caller
  dim count, loopid
  erase matches
  intdob = validDOB@datent(dob) ' convert dob
' is the date of birth in the index?
  if not exists(^PersonI("DOB", intdob)) then
    print "...no matches"
    exit sub
  end if
  loopid = ""
' loop through ids, and number them
  do
    count = count + 1
    loopid = traverse(^PersonI("DOB", intdob, loopid))
    if loopid = "" then exit do
    matches(count) = loopid
    print count, ") "
    display(loopid, "line")
  loop
  pick(id)
end sub
```

6. 以下は、**getphone** サブルーチンです。検索のために 3 桁のエリア・コードを指定すると、演習で述べたバグが発生します。**Traverse()** は、“-” 文字ではなく、数としてこれを解釈するので、以下の最初の **Traverse()** は正しい結果を返しません。回避策として、最初の **Traverse()** の前の行で、エリア・コードに “-” を追加します。

```
private sub getphone(origph as %String, ByRef id As %Integer)
' perform phone lookup
' if user picks a name from the list, id is returned to the caller
  erase matches
  dim count, loopid, ph
  count = 0 ' assume no matches
  if (origph > 0 and origph < 999) then
    origph = origph & "-" ' change to a string instead of a number
  end if
  ' origph may be an exact match, if not, use traverse()
  ph = origph
  if not exists(^Personl("Phone", origph)) then
    ph = traverse(^Personl("Phone", origph), 1, loopid)
  else
    loopid = ^Personl("Phone", origph)
  end if
  ' loop through phone numbers, and number them,
  ' quit as soon as phone doesn't match original
  ' loopid holds the ONE id per phone number
  do
    count = count + 1
    if left(ph, len(origph)) <> origph then exit do
    matches(count) = loopid
    print count, " "
    display(loopid, "line")
    ph = traverse(^Personl("Phone", ph), 1, loopid)
  loop
  if not exists(matches) then ' were there matches?
    print "...no matches"
    exit sub
  end if
  pick(id)
end sub
```

7. 以下は、getname サブルーチンです。

```
private sub getname(name As %String, ByRef id As %Integer)
' perform name lookup
' if user picks a name from the list, id is returned to the caller
erase matches
dim count, loopid, origln, origfn, ln, fn
count = 0 ' assume no matches
origln = piece(name, ",", 1) : origfn = piece(name, ",", 2)
' origln may be an exact match, if not, advance using traverse()
ln = origln
if not exists(^Personl("Name", origln)) then
set ln = traverse(^Personl("Name", origln))
end if
' loop through last names
' quit as soon as last name doesn't match original
do
if (left(ln, len(origln)) <> origln) then exit do
' origfn may be an exact match, if not, advance using traverse()
fn = origfn
if (origfn = "") or not exists(^Personl("Name", ln, origfn)) then
fn = traverse(^Personl("Name", ln, origfn))
end if
' loop through first names
' quit as soon as first name doesn't match original, or is ""
do
if ((left(fn, len(origfn)) <> origfn) or (fn = "")) then exit do
loopid = ""
' loop through ids
do
loopid = traverse(^Personl("Name", ln, fn, loopid))
if (loopid = "") then exit do
count = count + 1
matches(count) = loopid
print count, ") "
display(loopid, "line")
loop
fn = traverse(^Personl("Name", ln, fn))
loop
ln = traverse(^Personl("Name", ln))
loop
if not exists(matches) then ' were there matches?
print "...no matches"
exit sub
end if
pick(id)
end sub
```

8. 以下は、**display** サブルーチンです。

```
private sub display(id As %Integer, style as %String)
' given an id, get data and write it
dim rec
rec = ^PersonD(id)
name = piece(rec, "^", 1)
phone = piece(rec, "^", 2)
intdob = piece(rec, "^", 3)
if style = "line" then
    println name, space(2), phone, space(2), DateConvert(intdob,
vbToExternal)
else
    display@myBASdatent()
end if
end sub
```

9. 以下は、**pick** サブルーチンです。

```
private sub pick(ByRef id As %Integer)
' choose from the displayed items, and set up id
' id is 0 if no choice is made, id is >0 when user makes a choice
dim choice
do
    input "Choose by number: ", choice : println
    if (choice = "") then exit sub
    id = matches(choice)
    if (id <> "") then exit sub ' valid choice
    id = 0
    println "Invalid choice"
loop
end sub
```

10. [ファイル]→[保存] をクリックします。

11. ファイル名に *myBASlookup.BAS* を指定します。[名前を付けて保存] をクリックします。

12. Caché ターミナルを開始し、*SAMPLES* ネームスペースで `do main^myBASlookup()` と入力してルーチンを実行します。

Appendix G:

演習 4 : 編集機能の追加

1. 以下は、最終的な **myBASlookup** ルーチンです。わかりやすいように、セクションを分割して表示しています。このルーチンも *BASlookup2.mac* として *SAMPLES* ネットワークスペースで利用できます。
2. 以下は、変更した **main** セクションです。

```
public sub main()
  dim done
  do
    getsubmit(id, done) ' let user submit a string for lookup
    if id = 0 then continue do
    display(id, "table") ' display the chosen person
    edit(id) ' edit the chosen person
  loop until done
end sub
```

3. 以下は、新規の **edit** サブルーチンです。

```
private sub edit(id as %Integer)
' allow user to choose, and edit their choice
  dim yn, newname, newphone, newintdob
  do
    input "Edit? (y/n): ", yn : println
    if yn <> "y" then
      print "...no changes."
      exit sub
    end if
    ' try to lock the record
    if lock(^PersonD(id), 5) then exit do
    println "...someone else is editing this person"
  loop
  load(id)
  if reprompt(newname, newphone, newintdob) then
    input "Store? (y/n): ", yn : println
    if yn <> "y" then
      print "...no changes."
    else
      update(id, newname, newphone, newintdob)
    end if
  end if
  unlock(^PersonD(id))
end sub
```

4. 以下は、新規の `load` サブルーチンです。

```
private sub load(id as %Integer)
' load a person into local variables
  dim rec
  rec = ^PersonD(id)
  name = piece(rec, "^", 1)
  phone = piece(rec, "^", 2)
  intdob = piece(rec, "^", 3)
end sub
```

5. 以下は更新した `display` サブルーチンで、`load` を呼び出しています。

```
private sub display(id As %Integer, style as %String)
' given an id, get data and write it
  load(id)
  if style = "line" then
    println name, space(2), phone, space(2), DateConvert(intdob,
vbToExternal)
  else
    display@BASdatent2()
  end if
end sub
```

6. 以下は、新規の `reprompt` 関数です。

```
private function reprompt(ByRef newname As %String, _
                        ByRef newphone As %String, _
                        ByRef newintdob As %Integer) As %Boolean
' show current data and allow user to update it
  dim changed, newdob
  changed = False
  do
    print "Name: ", name, "=> "
    input newname : println
    if (newname = "") then
      newname = name ' default
    else
      newname = validName@BASdatent2(newname)
    end if
  loop while newname = 0
  if name <> newname then changed = True
  do
    print "Phone (617): ", phone, "=> "
    input newphone : println
    if (newphone = "") then
      newphone = phone ' default
    else
      newphone = validPhone@BASdatent2(newphone)
    end if
```

```

loop while newphone = 0
if phone <> newphone then changed = True
do
    print "DOB: ", DateConvert(intdob, vbToExternal), "=> "
    input newdob : println
    if (newdob = "") then
        newdob = DateConvert(intdob, vbToExternal) ' default
        newintdob = intdob
    else
        newintdob = validDOB@BASdatent2(newdob)
    end if
loop while newintdob = 0
if intdob <> newintdob then changed = True
println : println
return changed
end function

```

7. 以下は、新規の **update** サブルーチンです。

```

private sub update(id As %Integer, _
                    newname As %String, _
                    newphone As %String, _
                    newintdob As %Integer)
' update ^PersonD and ^PersonI
dim ln, fn, nln, nfn
tstart ' start a transaction
^PersonD(id) = newname & "^" & newphone & "^" & newintdob ' store the
record
if newname <> name then ' erase old name and add new name to index
    ln = piece(name, ",", 1) : fn = piece(name, ",", 2)
    nln = piece(newname, ",", 1) : nfn = piece(newname, ",", 2)
    erase ^PersonI("Name", ln, fn, id)
    ^PersonI("Name", nln, nfn, id) = ""
end if
if newphone <> phone then ' erase old phone and add new phone to index
    erase ^PersonI("Phone", phone)
    ^PersonI("Phone", newphone) = id
end if
if newintdob <> intdob then ' erase old dob and add new dob to index
    erase ^PersonI("DOB", intdob, id)
    ^PersonI("DOB", newintdob, id) = ""
end if
tcommit ' commit the transaction
println "...updated."
end sub

```

Appendix H:

演習 5 : データ入力と検索のオブジェクト・バージョンの記述

1. 以下は、**myBASdatentobj** の新規 **store** サブルーチンです。

```
public sub store()
' store the data
  dim yn, per

  input "Store? (y/n): ", yn ' see if user wants to store
' only go on if user says yes
  if ( yn <> "y" ) then
    println "...not stored."
    exit sub
  end if

  per = New BasTutorial.Person()
  with per
    .Name = name
    .Phone = phone
    .DOB = intdob
    .%Save()
  end with
  per = Nothing
  println "...stored" ' done
end sub
```

2. 以下は、**myBASlookupobj** の新規 **load** サブルーチンです。

```
private sub load(id as %Integer)
' load a person into local variables
  dim per
  per = OpenId BasTutorial.Person(id)
  with per
    name = .Name
    phone = .Phone
    intdob = .DOB
  end with
  per = Nothing
end sub
```

3. 以下は、**myBASlookupobj** の新規 **update** サブルーチンです。

```
private sub update(id As %Integer, _
                  newname As %String, _
                  newphone As %String, _
                  newintdob As %Integer)
' update ^PersonD and ^PersonI
  dim per
  per = OpenId BasTutorial.Person(id)
  with per
    .Name = newname
    .Phone = newphone
    .DOB = newintdob
    .%Save() ' this updates indexes too, all within a transaction!
  end with
  per = Nothing
  println "...updated."
end sub
```

4. この行を変更して、**myBASlookupobj** の **getphone** サブルーチンを更新します。

```
ph = traverse(^PersonI("Phone", origph), 1, loopid)
```

以下のとおり変更します。

```
ph = traverse(^PersonI("Phone", origph))
if ph <> "" then loopid = traverse(^PersonI("Phone", ph, ""))
```

5. 次にこの行を変更します。

```
loopid = ^PersonI("Phone", origph)
```

以下のとおり変更します。

```
loopid = traverse(^PersonI("Phone", origph, ""))
```

6. 次にこの行を変更します。

```
ph = traverse(^PersonI("Phone", ph), 1, loopid)
```

以下のとおり変更します。

```
ph = traverse(^PersonI("Phone", ph))
if ph <> "" then loopid = traverse(^PersonI("Phone", ph, ""))
```