

AutoCAD のための VBA の基礎、パート 5 - 2007 年 1 月

前回の記事では、良いコードの定義、一般的なコーディング ガイドラインの確認、マクロ構造の調査、コード編集用の強化機能(マイクロソフトのインテリジェンス機能およびコード補完機能)の紹介、便利なマクロ作成の実践といった、マクロの初歩的な概念を学びました。今回は前回触れた(例えば、定数、変数、オブジェクトやそれらの条件や制御) 概念のいくつかを説明することからはじめましょう。

ビジュアルベーシック言語について学びましょう

変数とは何でしょう

変数とは固有の名前でデータを保持するための容器です。それは、あなたが割り当てる価値を持つことができ、あなたのコードの中でオブジェクトの別名(短縮名) 役として働きます。あなたのプロジェクトやマクロの中で、変化する他のデータにとって代わるものとみなしてください。変数はマクロが走っている間だけ一時的に値を保持します。変数は名前(入れられる値の意味が分かりやすいもの) とデータ型(格納することが出来るデータの種類を決定) を持ちます。変数をあなたが良く知っている何かの容器と比較してみましょう。あなたが食料雑貨をお店から家まで運びたいとき、紙袋が欲しくなりますね。物を入れる代表は紙袋以外に何かありません。あなたは弁当を紙袋に入れて職場に持って行きますよね。薬局で買ったビタミンの瓶を家に運ぶときにも同じ紙袋を使い、12月の寒い夜、砂に挿したキャンドルを運ぶのにも紙袋は使われます。あなたはどんな袋に入れるか気にしますか。いいえ、あなたは単に後でそれから取り出したい物を入れるためだけに袋を使いますね。同じように、あなたは何かを運ぶためにバケツを使います。そして全く別の場合にも同じバケツを使うでしょう。バケツには、ピクニックに行く時にフライドチキンを入れたり、浜辺で砂のお城を作る時、家や車の窓を洗うための石鹸水を入れたりするかもしれません。容器について重要なのは運んでいる中身ではなく、異なる種類の物を運ぶことができるという事実です。あなたは目前の作業の中でどんな種類の容器を使うのが最適か経験を通じて学びます。同じように、VBA は、あなたが望むものを何でも格納できる容器(変数) を単に手元の作業に最適な容器として利用します。VBA には、また、あなたの使いたいアイテムがどんな型だかよくわからないときの為に、なんでも OK の大型旅行カバンのような変数が用意されています。この変数はバリエーション型と呼ばれていて、単語、文字列、数、物、など多種の物を格納するのに用いることができます。あなたの考えていることがわかります。「やった! いつでもバリエーション型を使えばいいのさあ〜」って。ついこのようにやりたくなるのはわかりますが、やらないでください。高い代償を払うことになりますよ。バリエーション型は実質的にあなたが求めるどんな物も変数に格納することを保証しますが、メモリを最大に消費します。初心者の間は特に誘惑に駆られますが、あなたがどの種類のデータを格納する為の変数を、どのデータ型にするか良く考えて適切に変数宣言するこ

とは、とても重要なことです。例えば、スーツケースは図書館から本を家に運ぶだけでなく、食料雑貨も運べます。しかしそれが多くの種類のアイテムを運ぶことが出来るからといって、全ての場合にあなたがそれを利用すべきであるということにはなりません。スーツケースが一週間にわたる旅行中に必要な全ての服を運ぶにふさわしい容器ではあっても、あなたの昼食を運ぶ為に使うには最適ではありません。解りましたか？

データ型とは何でしょう

データ型は、単にあなたの変数が持っているアイテムのタイプを記述しているカテゴリーです。例えば、紙袋に入ったあなたのお弁当は食べ物タイプですが、類似したものであり食べられるビタミンは、正確には食べ物タイプではなく別の似た物のタイプという事になります。変数のデータ型は、それらの値がどのようにコンピュータのメモリに保管されるかを決定します。あなたが変数を宣言する時、それにデータ型を指定することもできます。データ型の例は、文字列、整数、長さなどです。データ型は変数以外のものにも当てはまります。あなたが値をプロパティに割り当てる時その値はデータ型を持っています；関数に対する引数もデータ型を持っています。事実、VBA のデータのおよそ何でもがデータ型を持っています。変数はオブジェクトを代入することもできます。例えば、ユーザフォームやテキストボックスなどのオブジェクトです。

異なる状況の為に、多くの種類の変数タイプがあります、以下の表は一般的な変数の、データ型とそれらのおよそのメモリの占有容量のリストです。

| 名称 | 説明 | 内容 | 占有容量 |
|-------------|-----------------------------|--------------------------------------|----------|
| ブーリアン | 二者択一の一つの値 | 真または偽 | 2 バイト |
| バイト | 最小単位の亜類型 | 0 ~ 255 までの正数 | 1 バイト |
| カレンシー (通貨型) | 特別な数のフォーマット | 通貨 | 8 バイト |
| データ (日付型) | 特別な数のフォーマット | 日付または時間 | 8 バイト |
| ダブル型 | (倍精度浮動小数点数型) 倍精度数字フォーマット | 浮動小数点数 | 8 バイト |
| 整数型 | 整数 | -32768~32767 までの整数 | 2 バイト |
| 長整数型 | 正か負の整数 | -2,147,483,648 ~ 2,147,483,647 までの整数 | 4 バイト |
| 文字列型 | 単語または文字 | 昔々.. | 10+文字列 |
| バリエーション型 | いろいろ変わるデータ型 | 文字、数字、オブジェクト等 | 16 バイト以上 |

変数を宣言することは、あなたが前もって変数についてマクロに説明するという事です。これが良いコーディング練習を意味するので、我々は常にその使用に先立って変数を宣言するよう努めます。変数のもう一つの面は、変数の範囲を定義することです。範囲は、その変数が価値を持つことができるプロジェクトの領域として定義されます。たとえば、あなたがある変数はひとつのサブルーチンでのみ有効であると宣言するならば、そのサブルーチンの範囲以外では値をその変数に入れることができません。変数は、以下の範囲を持つことができます。

| 範囲 | 宣言 | 参照 |
|--------------|--------------------------------------|--------------------------|
| プロシジャーレベル | sub プロシジャー, function プロシジャー | 宣言されたプロシジャーのみ |
| モジュールレベル | フォームまたはコードモジュールの宣言セクション (.frm, .bas) | 宣言されたモジュール内のフォームや各プロシジャー |
| グローバル(パブリック) | コードモジュールの宣言セクション (.bas) | アプリケーション全体 |

一般的に Dim ステートメントで変数宣言は行われ、変数に名前をつけます。

(もしあなたがパブリック変数を作りたいならば Dim ステートメントの変数宣言文をパブリックステートメントに移動するだけです。)

タイプとしての Dim Variablename

あなたのプロジェクト全体で、データを形で割り当てる他の方法がない時だけ、グローバル変数を使用してください。グローバル変数を使わなければならないとき、単一のモジュールで全ての変数を宣言することは良いプログラミング練習です。それはファンクションステートメントでグループ化されます

前もって変数宣言をすることで得られる利益は、以下を含みます：誤植に起因するバグの数を減らして、マクロの実行時に占有されるメモリ量を最小にすることによってプログラミングの動作時間を節約すること。全てのフォーム、モジュール、クラス、その他で Option Explicit (変数宣言を強制) オプションを使うことを議論したのを思い出しましたか？

定数の使い方

定数は、不変の数または文字列に代わる意味がある名前です。定数は変数に似ていますが、定数は変数として動作している時も新しい値を代入したり、編集したりはできません。コードが変わることなく何度も繰り返し使われる値を含むとがわかります。また、あなたは、覚えるには困難なある数にコードが依存するのを理解するでしょう。その数は、終始数そのもので、明白な意味を持っていません。定数の一つの例は、pi 値 です。あなたはあなたのコードの読みや

すさを大きく改善できるでしょうし、定数を使うことでコードの保守がより簡単になるでしょう。

定数を宣言するための構文:

```
[Public|Private] Const constantname[As type] = expression
```

制御構造 (Control Structures) とは何でしょう

制御構造によってプログラム処理の流れをコントロールすることができます。過去には、プログラムは各々の実行される順番通りに記述されなければなりませんでした。もし同じ命令を実行したいならば、単に複数回その命令を書かなければなりませんでした。VBA では、他の高度な言語と同様に、制御文を作成することができます。何度も同じ命令を繰り返し書くことなくステートメントを繰り返すことができます。

非常に単純なプログラムをこの（左から右に展開する）トップダウンに示しますが、どんなプログラミング言語でもその大部分の力と有用性は、実行命令を条件付きの制御機構とループと交換するその能力によって決まります。

Control Structures の 2 つの基本タイプがあります：条件分岐と繰り返し構造です。

条件分岐構造

条件分岐制御の VBA 手順では、条件をテストすることができ、そしてそのテストの結果に従い、異なる処理を実行することができます。VBA がサポートする条件分岐構造：

If... Then

The simplest of Branching structures, the If Then structure can test a simple condition such as color of entity, or value of something. This is used in the following manner:

```
If some expression Then
    DO SOMETHING HERE
End if
```

If... Then... Else

A more complex version of the "If... Then" structure; provides the ability to do one thing if the expression is true; otherwise do something else.

Note: This doesn't mean that the second condition is true, just that the first

condition is not true so this is the only other possibility. This is used in the following manner:

If *some expression* **Then**

DO SOMETHING HERE

Else

DO SOME OTHER THING HERE

End if

Select Case

The most complex of the branching structures, it provides the ability to define many different conditions and include a default condition if none of the conditions is found to be true. This is used in the following manner:

Select Case *some value*

Case *First Value*

DO SOMETHING HERE

Case *Second Value*

DO SOMETHING HERE

Case *nth Value*

DO SOMETHING HERE

Case Else (Note: *This is optional*)

DO DEFAULT THING HERE

End Select

ループ（繰り返し）構造

ループ構造を利用することで、繰り返してそのコードを再タイプすることなく、一つまたは複数行のコードを繰り返し実行することができます。この反復ループは、長くも短くも要求された間だけ実行され、コード自身に記述された変数に従って変化します。VBA がサポートするループ構造：

Do... Loop

A Do loop executes a block of statements an indefinite number of times. There are several variations of the Do... Loop statement, but each type checks a numeric value before it continues the next loop. (Note: The statements never execute if condition is initially False.) As with If... Then, the condition

must be a value or expression that evaluates to False (zero) or to True (nonzero). This is used in the following manner:

Do While *condition to be checked*

DO SOMETHING HERE

Loop

For...Next

A For Next loop is the workhorse method for looping a known number of loops. It is frequently used for user defined collections and intrinsic VBA collections alike. Again there are a couple variations of the For Next Loop. We will show examples of both types since they are used so frequently. This is used in the following manner:

For counter = start To end [Step stepvalue]

DO SOMETHING HERE

Exit For

Next counter

This type can be used to loop through all the values contained in an array or for all values contained in a user-defined custom type of object (more on this later). The start is the minimum value for the loop, the counter is typically a variable which is incremented (by the stepvalue) until it reaches the end or maximum value. The Exit For command is used when the correct value is obtained since it is the only way to continue past the end of the loop without finishing the looping. (Note: You can also count backwards by reversing the start and end values and making your stepvalue equal some negative integer.)

For Each...Next

A For Each Next loop is the workhorse method for using built-in collections in AutoCAD's object model. It is frequently used when the count of the collection is unknown. A For Each...Next loop is similar to a For...Next loop, but it repeats a group of statements for each element in a collection of objects or in an array instead of repeating the statements a specified number of times. This is used in the following manner:

For Each Collection Item In Collection

DO SOMETHING HERE

Exit For

Next Collection Item

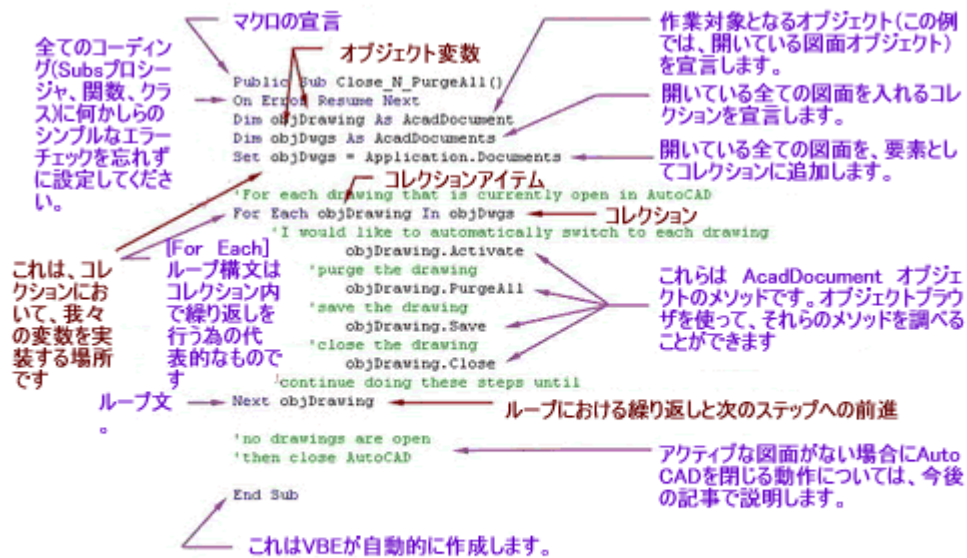
This type can also be used to loop through all the values contained in an array or for all values contained in a user-defined custom type of object (more on this later). The Exit For command is used when the correct value is obtained since it is the only way to continue past the end of the loop without finishing the looping. (Note: You usually have to simply declare the collection item but declare and populate the collection object.)

コレクションを利用しましょう

オブジェクトのコレクションを定めて使う能力は、VBA と AutoCAD の強力な特徴です。コレクションが有効な間に、最高のパフォーマンスを発揮するためにそれらを正しく使うべきです。コレクションは増加式カウンターによる For...Next ループの繰り返しを受け入れます。しかしほとんどの場合において、For Each...Next 構文が読みやすく維持が簡単でより早いです。For Each...NEXT による繰り返しは、コレクションに標準装備されたメソッドですが、コレクションそのものの作成者によって実行されますので、実際の速度は異なるかもしれません。

同じタイプのオブジェクトのグループがあるときは、通常コレクションまたは配列でそれらを管理することができます。処理スピードについては、あなたがオブジェクトにどのようにアクセスするかあなたが選ぶアプローチ次第です。コレクションの対象の数が 100 を超え、ユニークなキーで個々のオブジェクトを連想することができるならば、コレクションは最も速い選択です。あるオブジェクトをコレクションから取り戻すためにキーを使うことは、順番に配列を通して繰り返すことよりたいてい速いです。しかしながら、あなたがユニークなキーを持っていない為に、終わりまで配列を常に繰り返さなければならないならば、それはより良い選択です。オブジェクトが少数（100 足らず）ならば、配列はより少ないメモリ消費で、通常より速く検索することができます。

さあ、最後の記事から我々のコードをもう一度見てみましょう。挿絵を見ましょう。私たちがこの記事で学んだ新しい定義を利用し、それらがどこにどのように使われているかを示す為に既存の図にワイン色でいくつかのメモを加えてみました。



注意:我々はそれらをコレクションに実装したのではなく、コレクションそのものです。

今回は引き続き注目し、このルーチンをより知的により効率的にして、いくらかのエラーチェックを加えて AutoCAD と VBA で我々が利用できるように組み込み関数のいくつかを調査して利用できるようにしましょう。



(AUGI のディスカッション フォーラムでこの記事について討論しましょう！)



AUGI 理事会の会長であると同時に、ローカル ユーザ グループのマネージャでもあり、またフォーラム マネージャでもある Richard Binning による投稿。さらに Richard は AEC/IS Roundtable のメンバーであり、CAD、BIM、および技術関連のブログ Beside The Cursor の執筆者でもあります。Richard は、現在、The Haskell Company に AE テクノロジー アプリケーション マネージャとして勤務し、主要なオートデスク トレーニング センターである Technology Institute of the South のオートデスク認定インストラクタでもあります。